

Universitatea Politehnica București
Facultatea de Automatică și Calculatoare

Ecuatia undelor pentru acustica 2D

Autor:

Trăscău Mihai

Coordonatori Științifici:

ing. Mihai Victor Pricop

șl. dr. ing. Emil Slușanschi

București

Iulie 2009

Capitolul 1

Introducere

În decursul ultimilor ani nevoia de a realiza simulări computaționale ale unor procese fizice a devenit necesară, chiar indispensabilă, având în vedere faptul că dimensiunea și complexitatea problemelor abordate de diferitele ramuri științifice și industriale au crescut din ce în ce mai mult, costurile obținerii rezultatelor folosind metode experimentale au devenit prohibitive. Progresele recente remarcabile în domeniul High-Performance Computing, atât din punct de vedere al arhitecturii hardware, cât și din punct de vedere al instrumentelor software, au permis crearea unor aplicații deosebit de complexe și de mare acuratețe, capabile să ofere soluții corecte și complete pentru majoritatea nevoilor de simulare.

Una din ramurile care face uz de potențialul computațional al supercalculatoarelor și a sistemelor de tip grid este CFD (Computational Fluid Dynamics) care oferă soluții numerice pentru probleme de dinamica fluidelor. Fundamentul majorității problemelor de CFD se reduce la ecuațiile Navier-Stokes care descriu curgeri de fluide. Simplificarea acestor ecuații, prin eliminarea termenilor care descriu vâscozitatea și conducția termică, duce la ecuațiile lui Euler. Simplificări suplimentare aduc ecuațiile la forma liniarizată pentru potențiale, care pot fi folosite și pentru simulări de alte tipuri (cum ar fi cele acustice).

Un domeniu înrudit este și acustica, știință interdisciplinară care studiază sunetele, ultrasunetele și infrasunetele, toate reprezentând, de fapt, propagări ale unor unde mecanice prin gaze, lichide sau solide. În fluide precum aerul sau apa, undele acustice se propagă sub forma unor perturbații ale valorilor presiunii din mediu. Cu toate că, de obicei, aceste perturbații sunt de amplitudine mică, ele rămân detectabile de urechea umană, și sunt măsurate folosind o scară logaritmică în decibeli. În termeni fizici, pe lângă nivelul decibelilor, în propagarea sunetului este luată în calcul și frecvența sunetului (numărul de cicli pe secundă), ea fiind des folosită în analizele fonice.

În această lucrare se descrie implementarea unei aplicații paralele și distribuite de simulare a propagării undelor acustice folosind metode numerice de rezolvare. Aplicația folosește pentru rezolvare metoda diferențelor finite (MDF) calculată cu schema explicită, pentru care s-a realizat și analiza stabilității numerice. Modelul implementat este folosit pentru simularea unui domeniu 2D, pentru un fluid omogen, incompresibil și izotrop.

Pentru implementarea paralelismului și a calculului distribuit, algoritmi de rezolvare au fost construiți fie folosind interfața de programare OpenMP (Open Multi-Processing), fie folosind librăriile OpenMPI (Open Message Passing Interface) pentru protocolul de comunicație MPI, fie un hibrid între cele două. În scop didactic și pentru evaluarea performanțelor a fost implementat și varianta "serială" a aplicației.

Datorită faptului că aplicația a fost creată cu scopul de a face parte dintr-un pachet de aplicații și algoritmi de simulare pentru probleme de CFD, a fost necesară uniformizarea formatului și metodei de preluare a datelor de intrare. De asemenea, tot pe același principiu s-a implementat, ținând cont de celelalte aplicații, formatul și metoda de salvare a rezultatelor, unde s-a folosit formatul grafic VTK (Visualization Tool-Kit) care este independent de platformă și suportă randare paralelă.

Capitolul 2

Aspecte teoretice

2.1 Noțiuni generale fizice

Generarea undei acustice se datorează apariției unei perturbații locale pe fondul presiunii staționare a aerului. Această perturbație se propagă uniform în toate direcțiile influențând progresiv regiunile aflate pe direcția razei cu originea în punctul de generare a perturbației. Pentru a putea determina ecuația undei acustice se vor prezenta anumite aspecte geometrice preliminare, ce descriu propagarea perturbației.

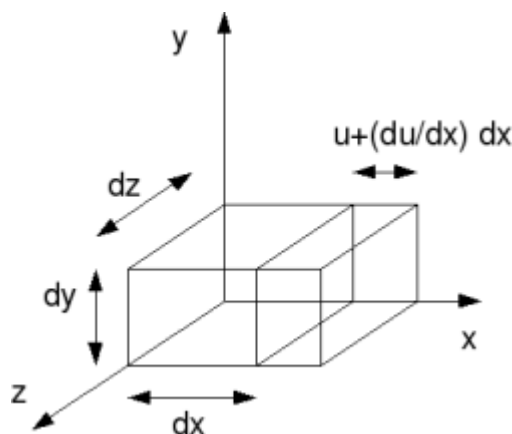


Figura 2.1: Propagarea perturbației

Vom defini volumul elementar:

$$dV = dx \cdot dy \cdot dz$$

$$\Delta(dV) = dx \frac{\partial u}{\partial x} \cdot dy \cdot dz$$

$$\varepsilon_V = \frac{\Delta(dV)}{dV} = \frac{\partial u}{\partial x} \quad (2.1)$$

Dacă vom considera că masa elementului rămâne constantă va rezulta că densitatea sa va suferi o modificare ca efect al modificării volumului.

$$\begin{aligned} dm &= \rho_0 dV = \rho(dV + \Delta(dV)) \\ (p - p_0)dV &= \rho \cdot \Delta(dV) \Rightarrow \varepsilon_V = \frac{p - p_0}{\rho} \end{aligned}$$

Se introduce noțiunea de condensare:

$$S = \frac{\rho - \rho_0}{\rho_0} \quad (2.2)$$

Se impune condiția de invarianță pentru masa elementului atât anterior deformației cât și după aceasta

$$(1 + \varepsilon_V)(1 + S) = 1 \Rightarrow \varepsilon_V + S + \varepsilon_V \cdot S = 0, \text{ dar } \varepsilon \cdot S \ll (S, \varepsilon_V) \Rightarrow S = -\varepsilon_V \quad (2.3)$$

Relația (3) prezintă corespondența între variația relativă a densității locale și modificarea din punct de vedere geometric suferită de volum, ca urmare a perturbației apărute de-a lungul axei O_x . Pentru a putea deduce ecuația undei acustice se vor introduce trei aspecte fizice ale fenomenului de propagare a perturbației de presiune.

Ecuația gazelor

Pentru simplificarea calculelor și pentru o mai bună înțelegere a conceptelor se vor introduce următoarele ipoteze simplificatoare:

- mediul în care unda acustică este generată și în care aceasta se propagă este un mediu gazos omogen și izotrop.
- fenomenul are caracter adiabatic, perturbația de presiune p adăugându-se presiunii staționare p_0 și având o valoare mult mai mică în comparație cu aceasta. Astfel se va considera presiunea mediului perturbat ca sumă a acestor două presiuni: $p_1 = p + p_0$.

$$p_1 V^\gamma = \text{constant}, \gamma = \frac{C_p}{C_V} \quad (2.4)$$

Prin diferențierea relației (4) se obține

$$dp_1 V^\gamma + \gamma p_1 V^{\gamma-1} dV = 0 \Rightarrow \frac{dp_1}{p_1} = -\gamma \frac{dV}{V} \quad (2.5)$$

Ținând cont de ipoteze se va nota:

$$dp_1 \cong p_1 - p_0 = p \quad p_1 = p_0 + p \cong p_0 \quad \Rightarrow \quad \frac{p}{p_0} = -\gamma \frac{dV}{V} \quad (2.6)$$

Considerând V ca fiind volumul elementar (notat anterior cu dV) și $dV = \Delta(dV)$ și ținând seama de ecuațiile (3) și (6) \Rightarrow

$$\frac{p}{p_0} = \gamma S \quad (2.7)$$

$$\frac{p}{p_0} = \gamma \frac{\rho - \rho_0}{\rho_0} \quad \Rightarrow \quad p = \gamma \frac{p_0}{\rho_0} (\rho - \rho_0) \quad (2.8)$$

Notăm cu c viteza undei de presiune în gazul considerat:

$$c^2 = \gamma \left(\frac{p_0}{\rho_0} \right) \quad \Rightarrow \quad (2.9)$$

$$p = c^2 (\rho - \rho_0) \quad (2.10)$$

Utilizând ecuația gazelor perfecte $c^2 = \gamma(\frac{RT}{\mu}) \Rightarrow$ viteza unei acustice poate fi scrisă ca o funcție depinzând de temperatură $v = v(T)$, iar p și ρ pot fi scrise ca funcții depinzând de poziție și de timp $p = p(x,t)$, $\rho = \rho(x,t)$. Derivând (10) în funcție de timp se obține

$$\frac{\partial p}{\partial t} = c^2 \frac{\partial \rho}{\partial t} \quad (2.11)$$

Ecuația de mișcare a elementului de masă corespunzător volumului elementar

$$\begin{aligned} v_x &= v_x(x, t) \\ dm \frac{\partial v_x}{\partial t} &= -\left(\frac{\partial p_1}{\partial x} \cdot dx\right) \cdot dy \cdot dz \\ dm &= \rho dx dy dz \cong \rho_0 dx dy dz \\ \frac{\partial p_1}{\partial x} &= \frac{\partial p}{\partial x} \end{aligned} \quad (2.12)$$

Ecuația (12) poate fi scrisă ca

$$\rho_0 dx dy dz \left(\frac{\partial v_x}{\partial t}\right) = -\left(\frac{\partial p}{\partial x}\right) dx dy dz \quad (2.13)$$

Pornind de la modelul ecuației (13) se scriu ecuațiile de mișcare pe cele 3 direcții:

$$-\frac{\partial p}{\partial x} = \rho_0 \frac{\partial v_x}{\partial t} \quad (2.14)$$

$$-\frac{\partial p}{\partial y} = \rho_0 \frac{\partial v_y}{\partial t} \quad (2.15)$$

$$-\frac{\partial p}{\partial z} = \rho_0 \frac{\partial v_z}{\partial t} \quad (2.16)$$

Ecuația de continuitate a ‘curgerii’ reprezentată de transmiterea perturbației

Vom considera debitul unitar al masei de fluid pe direcția $O_x q_m = \rho v_x$. Același model poate fi aplicat și pentru celelalte axe. Variația debitului în intervalul dt va contribui prin modificarea densității elementului de volum la modificarea masei acestuia.

$$dm = \left(\frac{\partial p}{\partial t}\right) dt dx dy dz$$

Dacă se consideră curgerea ca fiind concomitentă pe toate cele trei direcții ale axelor va rezulta:

$$\begin{aligned} \rho v_x dt dy dz - (\rho v_x + \frac{\partial(\rho v_x)}{\partial x} dx) dt dy dz + \rho v_y dt dz dx - (\rho v_y + \frac{\partial(\rho v_y)}{\partial y} dy) dt dz dx + \\ \rho v_z dt dx dy - (\rho v_z + \frac{\partial(\rho v_z)}{\partial z} dz) dt dx dy - \frac{\partial p}{\partial t} dt dx dy dz = 0 \end{aligned} \quad (2.17)$$

Dacă se fac reducerile corespunzătoare se obține

$$-\frac{\partial(\rho v_x)}{\partial x} - \frac{\partial(\rho v_y)}{\partial y} - \frac{\partial(\rho v_z)}{\partial z} = \frac{\partial p}{\partial t} \quad (2.18)$$

Astfel ținând cont de ipotezele referitoare la variația mică a densității în raport cu densitatea mediului neperturbat rezultă:

$$\frac{\partial(\rho v_x)}{\partial x} \cong \rho_0 \frac{\partial v_x}{\partial x} \quad (\text{similar pentru celelalte direcții}) \quad (2.19)$$

Relația (18) devine:

$$\rho_0 \left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z} \right) + \frac{\partial p}{\partial t} = 0 \quad (2.20)$$

2.1.1 Ecuația undei

Se poate stabili o relație de dependență a presiunii la un moment dat, într-un punct al mediului perturbat pe baza ecuațiilor (11) (14) și (20) și ținând seama de funcția $p = p(t, x, y, z)$. De asemenea este necesară o derivare suplimentară cu t, x, y, z a acestor ecuații:

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} + \frac{\partial^2 p}{\partial z^2} = \frac{1}{c^2} \cdot \frac{\partial^2 p}{\partial t^2} \quad (2.21)$$

Se introduc relațiile:

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} + \frac{\partial^2 p}{\partial z^2} = \Delta p \quad \Delta p = \nabla^2 p \quad (2.22)$$

unde $\nabla = \frac{\partial}{\partial x} \vec{i} + \frac{\partial}{\partial y} \vec{j} + \frac{\partial}{\partial z} \vec{k}$

Relația (21) care reprezintă ecuația în derivate parțiale a undei de presiune mai poate fi scrisă

$$\nabla^2 p = \frac{q}{c^2} \frac{\partial^2 p}{\partial t^2} \quad (2.23)$$

Dacă se consideră că funcția de viteze poate constitui o altă posibilitate de reprezentare a undei de perturbație și se impune condiția ca unda de presiune să nu creeze vârtejuri în mediu (câmpul de viteze să fie potențial) atunci se pot scrie relațiile:

$$\vec{v} = \vec{v}(x, y, z, t) \quad \vec{v} = \text{grad}\Phi \quad \Phi = \Phi(x, y, z, t)$$

Prin prelucrarea ecuațiilor (11) (14) (20) și înlocuirea funcției p cu potențialul de viteze Φ se obține:

$$\nabla^2 \Phi = \frac{1}{c^2} \frac{\partial^2 \Phi}{\partial t^2} \quad (2.24)$$

Propagarea uniformă a undei pe toate cele trei direcții dacă se consideră punctul de producere a perturbațiilor în centrul unei sfere este exprimată prin relațiile (22) și (24). Aceste ecuații caracterizează undele sferice care în concordanță cu principiul lui Huygens au funcțiile p și Φ respectiv în fază pe o sferă cu centrul de producere a perturbației și raza r .

2.1.2 Unda plană

Dacă se consideră o distanță suficient de mare față de originea perturbației ($r \rightarrow \infty$) atunci curba sferei devine suficient de mică pentru a considera că suprafața de undă este plană pe o zonă restrânsă în jurul unui punct aparținând acesteia. Se va considera astfel o undă plană care se propagă în lungul axei O_x . În ecuația (24) se aleg:

$$v_x \neq 0, v_y = v_z = 0, \Phi = \Phi(x, t) \quad (2.25)$$

$$\frac{\partial^2 \Phi}{\partial x^2} = \frac{1}{c^2} \frac{\partial^2 \Phi}{\partial t^2} \quad (2.26)$$

Pentru a rezolva ecuația (26) există două metode. Prima metodă presupune apariția unei perturbații pe axa O_x care se transmite uniform în ambele sensuri ale axei cu viteza c , astfel încât unda în sens contrar axei O_x poate fi considerată unda reflectată, așa cum este descris și în figura (2).

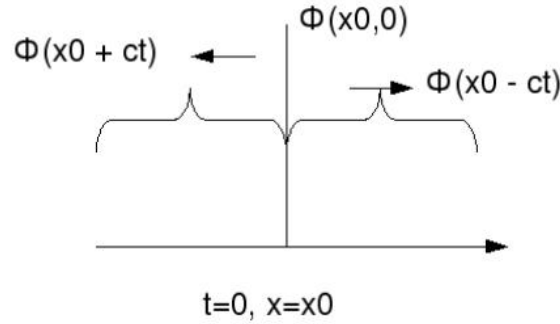


Figura 2.2: Unda propagată și cea reflectată

Pentru Φ se alege o soluție de forma:

$$\Phi(x, t) = f_1(x - ct) + f_2(x + ct) \quad (\text{verific ecuația (26)}) \quad (2.27)$$

O altă rezolvare pentru ecuația (26) poate fi în forma periodică alegând:

$$\Phi(x, t) = X(x) \cdot T(t) \quad (2.28)$$

Prin înlocuirea (28) în (26) și impunând $T(t)$ periodică de pulsație ω

$$X''T = \frac{1}{c^2}X\ddot{T} \Rightarrow c^2\frac{X''}{X} = \frac{\ddot{T}}{T} = -\omega^2 \quad (2.29)$$

Prin integrarea ecuației (29) se obține:

$$T = B \cos(\omega t - \rho_1) \quad X = C \cos\left(\frac{\omega}{c}x - \rho_2\right) \quad (2.30)$$

Introducând soluția obținută în relația (30) în relația (28) se obține:

$$\Phi(x, t) = A \cos(\omega t - \rho_1) \cdot \cos\left(\frac{\omega}{c}x - \rho_2\right) \quad (2.31)$$

În relația prezentă înlocuim produsul de cosinusuri prin suma:

$$\Phi(x, t) = A \left[\cos\left(\omega t - \frac{\omega}{c}x - \rho_A\right) + \cos\left(\omega t + \frac{\omega}{c}x - \rho_B\right) \right] \quad (2.32)$$

Φ este periodică atât după t cât și după $x \Rightarrow$

$$\begin{aligned} \omega(t + 2\pi) = \omega(t + T) &\Rightarrow T = \frac{2\pi}{\omega} \quad (\text{variație în timp}) \\ \frac{\omega}{c}x + 2\pi = \frac{\omega}{c}(x + \lambda) &\Rightarrow \lambda = \frac{2\pi}{\omega}c = Tc \quad (\text{variație în abscisa}) \end{aligned} \quad (2.33)$$

Se notează $k = \frac{\omega}{c}$

$$k = \frac{2\pi}{\lambda}$$

Dacă se neglijează ρ_A și ρ_B , neglijare ce nu influențează aspectul problemei, atunci Φ va avea următoarea formă, similară cu cea din expresia (27).

$$\Phi(x, t) = A [\cos(\omega t - kx) + \cos(\omega t + kx)] \quad (2.34)$$

Mărimi caracteristice undei plane

Pentru a exprima ecuația undei acustice plane se pot folosi formulele (22) sau (24), care în cazul unidimensional pot fi scrise:

$$\frac{\partial^2 p}{\partial x^2} = \frac{1}{c^2} \frac{\partial^2 p}{\partial t^2}; \quad \frac{\partial^2 \Phi}{\partial x^2} = \frac{1}{c^2} \frac{\partial^2 \Phi}{\partial t^2} \quad (2.35)$$

Mai departe se va considera rezolvarea cu forma p a ecuației și se va lua în considerare numai termenul undei directe

$$p = p \cos(\omega t - kx) \quad (2.36)$$

Din (14) \Rightarrow

$$-\frac{\partial p}{\partial x} = \rho_0 \frac{\partial v_x}{\partial t} \Rightarrow v_x = \frac{-1}{\rho_0} \int \frac{\partial p}{\partial x} dt = \frac{k}{\rho_0 \omega} p \sin(\omega t - kx) \quad (2.37)$$

Dacă se înlocuiește expresia corespunzătoare lui $k \Rightarrow$

$$v_x = \frac{p}{\rho_0 c} \sin(\omega t - kx) \quad (2.38)$$

Exprimări complexe

Pentru a rezolva ecuații diferențiale se consideră în general numai partea reală a soluției. Dacă se are în vedere o soluție periodică atunci se folosește sinus sau cosinus, corespunzător în planul complex poziției A a soluției pentru care $z_A = x_A + iy_A$. Coordonatele lui A sunt $x_A = \cos(\omega t)$, $y_A = \sin(\omega t)$ ceea ce corespunde în exprimarea cu modul și argument, a relației:

$$z_A = r^{i\omega t}, e^{i\omega t} = \cos(\omega t) + i \sin(\omega t)$$

Dacă se condensează relațiile (33) și (34)

$$\tilde{\Phi}(x, t) = A(e^{i(\omega t - kx)} + e^{i(\omega t + kx)})$$

Astfel se poate lucra cu $\Re(\tilde{\Phi})$ și $\Im(\tilde{\Phi})$. Această formă, mai generală fiind, poate descrie un front de undă oblic, spre deosebire de exprimarea în forma reală care nu permite acest lucru. Expresiile (36) și (38) definesc o mărime ce caracterizează propagarea într-un spațiu liber, infinit și fără reflexii a undei acustice. Această mărime poartă numele de impedanță acustică și se definește:

$$Z_s = \frac{p}{v_x} = \rho_0 c \quad \left[\frac{kg}{m^2 \cdot s} \right] \quad (2.39)$$

Din relația (39) rezultă că impedanța este o mărime reală. Ea poate fi însă prezentată și sub forma complexă, dacă se consideră incinte acustice cu forme deosebite.

Mediu	c (m/s)	ρ_0 ($\frac{kg}{m^3}$)
aer (20°)	344	1,2
ap[(13°)	1441	1000
oțel	5100	7800
sticlă	6000	2400

Figura 2.3: Viteze și densități

Densitatea de energie acustică se reprezintă ca sumă a energiilor cinetică și potențială, ce sunt conținute în unitatea de volum a frontului de undă pe direcția de propagare a acestuia.

$$dE_c = \frac{1}{2} \rho_0 dV v_x^2 \quad (E_c \text{ elementara})$$

Valoarea energiei cinetice raportată la volum se obține prin împărțirea cu dV . Energia potențială (E_p) specifică este definită ca fiind energia înmagazinată de elementul de volum ce se deformează sub acțiunea presiunii p :

$$W_d = \frac{1}{2} p \varepsilon_v = \frac{1}{2} \frac{p}{c^2 \rho_0}$$

$$E = \frac{dE_c}{dV} + W_d = \frac{1}{2} \rho_0 v_x^2 + \frac{1}{2} \frac{p}{c^2 \rho_0}$$

Se calculează utilizând ecuațiile (36) și (38) valorile pentru unda progresivă

$$E = \frac{1}{2} \rho_0 \frac{p^2}{\rho_0^2 c^2} \sin_2(\omega t - kx) + \frac{1}{2} \frac{p^2}{\rho_0 c^2} \sin_2(\omega t - kx) = \frac{p^2}{\rho_0^2 c^2} \sin_2(\omega t - kx)$$

Dacă se consideră o întreagă perioadă T se obține valoarea medie a energiei acustice:

$$\bar{E} = \frac{1}{T} \int_0^T E dt = \frac{p^2}{T \rho_0 c^2} \int_0^T \sin(\omega t - kx) dt = \frac{1}{2} \frac{p^2}{\rho_0 c^2}$$

$$\text{Notăm } p_{ef} = \frac{p}{\sqrt{2}} \Rightarrow \bar{E} = \frac{p_{ef}^2}{\rho_0 c^2}$$

Intensitatea acustică medie se definește ca fiind fluxul de energie acustică ce străbate frontul unei plane a perturbației cu viteza sunetului.

$$\bar{I} = \bar{E}c = \frac{p_{ef}^2}{\rho_0 c}$$

2.1.3 Unda sferică

Pentru a reprezenta propagarea perturbației locale a presiunii se pot folosi ecuațiile (22) și (24). Abordarea prezentată anterior, unda plană, reprezintă un caz particular ce se pretează când distanța față de sursa perturbației este mare, astfel încât planul tangent într-un punct al sferei poate fi asimilat cu o suprafață sferică în jurul acestui punct de tangență. Dacă se consideră că centrul sferei este originea sursei de perturbație atunci o abordare logică ar fi cea în coordonate sferice, legate de coordonatele carteziane prin relațiile (deduse din figura (4)):

$$x = r \sin \theta \cos \varphi$$

$$y = r \sin \theta \sin \varphi$$

$$z = r \cos \theta$$

Dacă se înlocuiesc aceste relații în relația (24), aceasta devine

$$\nabla^2 \Phi = \frac{\partial^2 \Phi}{\partial r^2} + \frac{2}{r} \frac{\partial \Phi}{\partial r} + \frac{1}{r^2 \sin \theta} \cdot \frac{\partial}{\partial \theta} (\sin \theta \frac{\partial \Phi}{\partial \theta}) + \frac{1}{r^2 \sin \theta} \cdot \frac{\partial^2 \Phi}{\partial \varphi^2} \quad (2.40)$$

Dacă spațiul de propagare se consideră omogen, izotrop și liber, se poate considera că Φ variază numai în funcție de r , acest lucru conducând la următoarele ecuații simplificate:

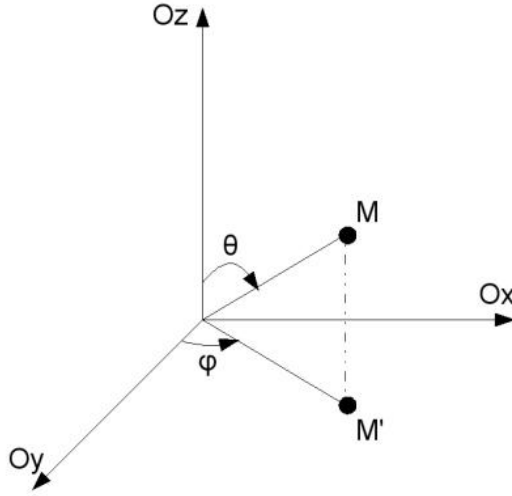


Figura 2.4: Coordonate sferice

$$\begin{aligned}\nabla^2\Phi &= \frac{\partial^2\Phi}{\partial r^2} + \frac{2}{r} \frac{\partial\Phi}{\partial r} = \frac{1}{r} \frac{\partial^2(r\Phi)}{\partial r^2} \\ \frac{1}{r} \frac{\partial^2(r\Phi)}{\partial r^2} &= \frac{1}{c^2} \frac{\partial^2\Phi}{\partial t^2} \\ \frac{\partial^2(r\Phi)}{\partial r^2} &= \frac{1}{c^2} \frac{\partial^2(r\Phi)}{\partial t^2}\end{aligned}\quad (2.41)$$

Ecuția (41) poate fi soluționată asemenea ecuației (26) înlocuind funcția Φ cu $r\Phi$ iar ecuația (22) poate fi scrisă în coordonate sferice astfel:

$$\frac{\partial^2(rp)}{\partial r^2} = \frac{1}{c^2} \frac{\partial^2(rp)}{\partial t^2} \quad (2.42)$$

Se integrează (42) sub forma complexă și se obține expresia unei progresive:

$$\tilde{p} = \frac{p}{r} e^{i(\omega t - kr)} \quad (2.43)$$

Dacă se dorește exprimarea vitezei atunci se utilizează prima ecuație din (14) în care p nu variază cu ω și φ

$$-\frac{\partial p}{\partial r} = \rho_0 \frac{\partial v_r}{\partial t} \quad (2.44)$$

Prin înlocuirea ecuației (43) în (44) se obține:

$$\frac{p}{r^2}e^{i(\omega t - kr)} + \frac{p}{r}e^{i(\omega t - kr)} = \rho_0 \frac{\partial v_r}{\partial t} \Rightarrow$$

$$\tilde{v}_r = \frac{p}{\rho_0} \left(\frac{1}{r^2} + \frac{ik}{r} \right) \int e^{i(\omega t - kr)} dt$$

Dacă legea de variație nu se modifică atunci se poate considera constanta de integrare ca fiind nulă:

$$\tilde{v}_r = \frac{p}{i\omega\rho_0} \left(\frac{1}{r^2} + \frac{ik}{r} \right) e^{i(\omega t - kr)} \quad (2.45)$$

Mărimi ce caracterizează unda acustică sferică

În ecuațiile (43) și (44) atât \tilde{p} cât și \tilde{v}_r au amplitudini și faze constante pe o suprafață sferică cu originea în centrul perturbației și raza r . Astfel este verificat principiul lui Huygens referitor la unde sferice și evoluția acestora în timp și spațiu. Impedanța acustică se prezintă în cazul undei sferice sub forma complexă, spre deosebire de exprimarea din relația (39) în care impedanța are o valoare reală.

$$\tilde{Z} = \frac{\tilde{p}}{\tilde{v}} = \frac{i\omega\rho_0 r}{1 + irk} = \rho_0 c \frac{ikr}{1 + ikr} \quad (2.46)$$

Dacă $r \rightarrow \infty$ sau $r \gg \lambda$ $kr = 2\pi \frac{r}{\lambda}$ atunci:

$$\tilde{Z} = \rho_0 c = Z_s$$

Intensitatea acustică medie se definește:

$$I = pv \Rightarrow \tilde{I} = \frac{1}{T} \int_0^T p v dt \quad (2.47)$$

Pentru p și v se va lua în considerare numai partea reală a expresiilor corespunzătoare

$$e^{i(\omega t - kr)} = \cos(\omega t - kr) + i \sin(\omega t - kr)$$

$$p = \Re(\tilde{p}) = \frac{p}{r} \cos(\omega t - kr)$$

$$v = \Re(\tilde{v}) = \frac{pk}{\omega\rho_0 r} \cos(\omega t - kr) + \frac{p}{\omega\rho_0 r^2} \sin(\omega t - kr)$$

Se calculează integrala (47):

$$\bar{I} = \frac{p^2 k}{2\omega \rho_0 r^2} = \frac{\left(\frac{p}{r}\right)^2}{2} \cdot \frac{1}{\rho_0 c} = \frac{(\tilde{P}_{ef})^2}{\rho_0 c}$$

Densitatea de energie acustică este definită ca:

$$\bar{E} = \frac{\bar{I}}{c} = \frac{(\tilde{P}_{ef})^2}{\rho_0 c}$$

Energia radiată de sursă reprezintă o valoare constantă prin fiecare suprafață sferică cu centru în sursă și se definește:

$$W = \bar{I} 4\pi r^2 = \frac{2\pi p^2}{\rho_0 c}$$

2.1.4 Reflexia

Se consideră o suprafață S, ce separă doua medii în care viteza de propagare a sunetului este diferită. Presupunem c_1 viteza de propagare a sunetului în mediul 1 și c_2 viteza de propagare a sunetului în mediul 2, și deasemenea că $c_1 \gg c_2$:

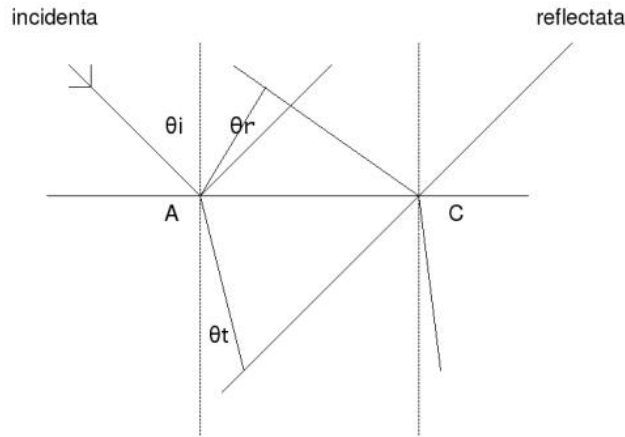


Figura 2.5: Cameră paralelipipedică

AC_1 și CA_1 sunt două fronturi perpendiculare pe razele de incidență și implicit pe cele reflectate de la suprafața S. Timpul în care este parcursă distanța CC_1 este același cu timpul necesar parcurgerii distanței AA_1

$$t_1 = \frac{CC_1}{c_1} = \frac{AA_1}{c_1} \Rightarrow CC_1 = AA_1$$

$$\triangle ACC_1 \equiv \triangle ACA_1 \Rightarrow \angle C_1AC \equiv \angle A_1CA \Rightarrow Q_i \equiv Q_r$$

Frontul celei de-a doua unde, refractată în mediul al doilea este CA_2 . Timpul de parcurgere a distanței CC_1 este identic cu cel în care raza transmisă în al doilea mediu parcurge AA_2

$$t_1 = \frac{C_1C}{c_1} = \frac{AA_2}{c_2}$$

Se notează

$$\begin{aligned} \angle ACA_2 = \Theta_t \quad \sin \Theta_t &= \frac{AA_2}{AC} \quad \sin \Theta_i = \frac{CC_1}{AC} \Rightarrow \\ AC &= \frac{CC_1}{\sin \Theta_i} = \frac{AA_2}{\sin \Theta_t} \Rightarrow \\ \frac{\sin \Theta_t}{\sin \Theta_i} &= \frac{c_2}{c_1} \end{aligned}$$

Dacă se ține seama de relația din ipoteza ce specifică $\Theta_t < \Theta_i$ și de faptul că $c_1 < c_2$ rezultă asemenea cazului luminii, posibilitatea reflexiei totale ($\Theta_t > 90^\circ$). Astfel unghiul de incidență Θ_i^* necesar pentru apariția fenomenului de reflexie totală se obține la $\Theta_t = 90^\circ \Rightarrow$

$$\sin \Theta_i^* = \frac{c_1}{c_2}$$

2.1.5 Acustica spațiilor închise. Unde acustice neamortizate

Modelul folosit este acela al unei camere paralelipedice, ce are pereții rigizi și perfect reflectanți iar sursa perturbației va fi plasată în originea axelor. Se vor considera l_x, l_y, l_z lungimile muchiilor paralelipedului în sensul celor trei axe.

În urma acestor ipoteze dacă unda acustică nu are desprindere la suprafața peretelui atunci viteza particulei de aer aflată în contact cu peretele este zero. Mai departe se va lucra cu exprimarea în coordonate carteziene, în funcție de potențial.

$$\frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} + \frac{\partial^2 \Phi}{\partial z^2} = \frac{1}{c^2} \frac{\partial^2 \Phi}{\partial t^2} \quad (2.48)$$

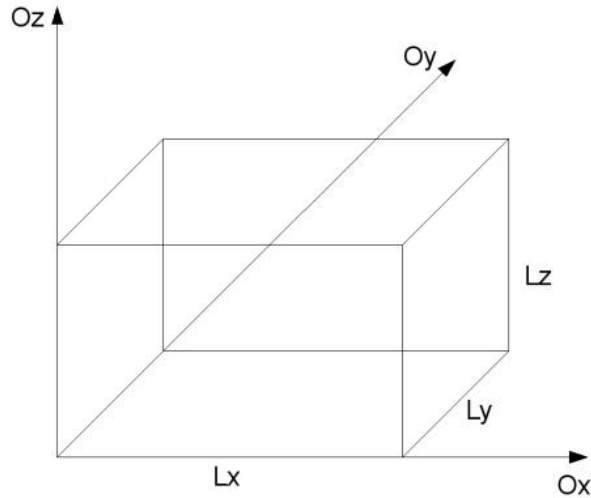


Figura 2.6: Cameră paralelipipedică

Pentru rezolvare se va folosi metoda separării variabilelor, expresia potențialului devenind:

$$\Phi(x, y, z, t) = X(x) \cdot Y(y) \cdot Z(z) \cdot T(t) \quad (2.49)$$

Întroducând relația (49) în (48) și împărțind expresia cu $xyzT$ se obține:

$$\frac{1}{x} \frac{d^2 X}{dx^2} + \frac{1}{y} \frac{d^2 Y}{dy^2} + \frac{1}{z} \frac{d^2 Z}{dz^2} = \frac{1}{c^2} \frac{1}{T} \frac{d^2 T}{dt^2} \quad (2.50)$$

În relația (49) se pune condiția de periodicitate în timp a semnalului acustic:

$$\frac{1}{c^2} \frac{1}{T} \frac{d^2 T}{dt^2} = -\frac{\omega^2}{c^2}$$

$$\ddot{T} + \omega^2 T = 0 \quad \Rightarrow \quad T = C_T \cos(\omega t - \psi_t) \quad (2.51)$$

Notând cu ω pulsația proprie a semnalului emis în O, iar C_T și ψ_t sunt constante de integrare. Fiecare dintre cei patru termeni ai ecuației (50) este independent de ceilalți, rezultând că pentru a ajunge la o sumă nulă, fiecare termen în parte trebuie să fie constant. Se consideră constant primul termen, corespunzător variației în timp. El este ales astfel încât să conducă la o oscilație. Procedul se repetă ținându-se seama la fiecare funcție de constantele introduse în pașii anteriori.

$$\begin{aligned}\frac{1}{Z} \frac{d^2 Z}{dz^2} &= -\frac{\omega^2}{c^2} + \mu^2 \\ Z'' + \left(\frac{\omega^2}{c^2} - \mu^2\right) Z &= 0 \\ Z &= C_Z \cos\left[\left(\sqrt{\frac{\omega^2}{c^2} - \mu^2}\right) Z - \psi_z\right] \quad (C_z, \psi_z \text{ constante de integrare})\end{aligned}\quad (2.52)$$

$$\frac{1}{Y} \frac{d^2 Y}{dy^2} = -v^2 \quad \Rightarrow \quad Y'' + v^2 Y = 0 \quad (2.53)$$

$$\begin{aligned}Y &= C_y \cos(v_y - \psi_y) \quad C_y, \psi_y \text{ constante de integrare} \\ \frac{1}{x} \frac{d^2 X}{dx^2} - v^2 - \frac{\omega^2}{c^2} + \mu^2 &= -\frac{\omega^2}{c^2} \\ x^2 + (\mu^2 - v^2)x &= 0 \\ x &= C_x \cos\left[\left(\sqrt{\mu^2 - v^2}\right)x - \psi_x\right] \quad C_x, \psi_x \text{ constante de integrare}\end{aligned}\quad (2.54)$$

Toate constantele fundamentale ω , μ , v cât și termenii în care acestea apar sunt alese de așa natură încat să conducă la mișcări periodice, atât în funcție de t , cât și de x , y , z . Reunind expresiile (51) ÷ (54) într-o expresie compactă corespunzătoare soluției (49) se obține:

$$\begin{aligned}\Phi(x, y, z, t) &= C \cos\left[\left(\sqrt{\mu^2 - v^2}\right)x - \psi_x\right] \cdot \cos(v_y - \psi_y) \cdot \\ &\cdot \cos\left[\left(\sqrt{\frac{\omega^2}{c^2} - \mu^2}\right)Z - \psi_z\right] \cdot \cos(\omega t - \psi_t)\end{aligned}\quad (2.55)$$

Pornind de la relația (55) ce reprezintă expresia potențialului de viteze, se determină constantele fundamentale ω , μ , v ce se obțin din condițiile impuse asupra vitezei particulei de aer la suprafața pereților incintei.

$$v_x = \frac{\partial \Phi}{\partial x}, \quad v_y = \frac{\partial \Phi}{\partial y}, \quad v_z = \frac{\partial \Phi}{\partial z} \quad (2.56)$$

Se pun condițiile

$$\{x = 0, y = 0, z = 0, v_x = v_y = v_z = 0, x = l_x, y = l_y, z = l_z\} \quad (2.57)$$

$$\Rightarrow \sin \psi_x = 0, \sin \psi_y = 0, \sin \psi_z = 0 \quad \psi_x = \psi_y = \psi_z = 0$$

$$\sin[(\sqrt{\mu^2 - v^2})l_x] = 0$$

$$\sin(v_y)l_y = 0 \Rightarrow$$

$$\sin[(\sqrt{\frac{\omega^2}{c^2} - \mu^2})l_z] = 0$$

$$\left\{ \begin{array}{l} \sqrt{\mu^2 - v^2} = \frac{n_x \pi}{l_x} \quad n_x = 1, \bar{n} \\ v = \frac{n_y \pi}{l_y} \quad n_y = 1, \bar{n} \\ \sqrt{\frac{\omega^2}{c^2} - \mu^2} = \frac{n_z \pi}{l_z} \quad n_z = 1, \bar{n} \end{array} \right. \quad (2.58)$$

$$\left\{ \begin{array}{l} v = \frac{n_y \pi}{l_y} \quad n_y = 1, \bar{n} \end{array} \right. \quad (2.59)$$

$$\left\{ \begin{array}{l} \sqrt{\frac{\omega^2}{c^2} - \mu^2} = \frac{n_z \pi}{l_z} \quad n_z = 1, \bar{n} \end{array} \right. \quad (2.60)$$

Rezolvând sistemul se obțin ω (pulsăția proprie) și f (frecvența proprie a camerei)

$$\omega = \pi c \sqrt{\left(\frac{n_x}{l_x}\right)^2 + \left(\frac{n_y}{l_y}\right)^2 + \left(\frac{n_z}{l_z}\right)^2} \quad (2.61)$$

$$f = \frac{c}{2} \sqrt{\left(\frac{n_x}{l_x}\right)^2 + \left(\frac{n_y}{l_y}\right)^2 + \left(\frac{n_z}{l_z}\right)^2} \quad (2.62)$$

Având în vedere că n_x, n_y, n_z pot lua orice valoare întreagă, rezultă că în afara pulsațiilor fundamentale corespunzătoare fiecărei direcții (ex: $n_x = 1, n_y = n_z = 0$) se pot obține o infinitate de armonice. Pe baza relației $p = -p_0 \frac{\partial \Phi}{\partial t}$ rezultă relația de variație a presiunii.

$$p = P \cos\left(\frac{n_x \pi}{l_x} x\right) \cos\left(\frac{n_y \pi}{l_y} y\right) \cos\left(\frac{n_z \pi}{l_z} z\right) e^{i\omega t} \quad (2.63)$$

Frecvențele proprii ale unei camere sunt în număr ininit așa cum se poate observa din relațiile de mai sus. Din relația (62) se observă că frecvența are proiecții pe cele trei axe formate dintr-un număr întreg de segmente de tipul:

$$\Delta f_x = \frac{c}{2l_x}, \quad \Delta f_y = \frac{c}{2l_y}, \quad \Delta f_z = \frac{c}{2l_z}$$

Astfel se creează o rețea spațială în care fiecărui punct nodal îi corespunde o frecvență de oscilație într-un plan normal la \vec{f} , definit pentru punctul nodal respectiv:

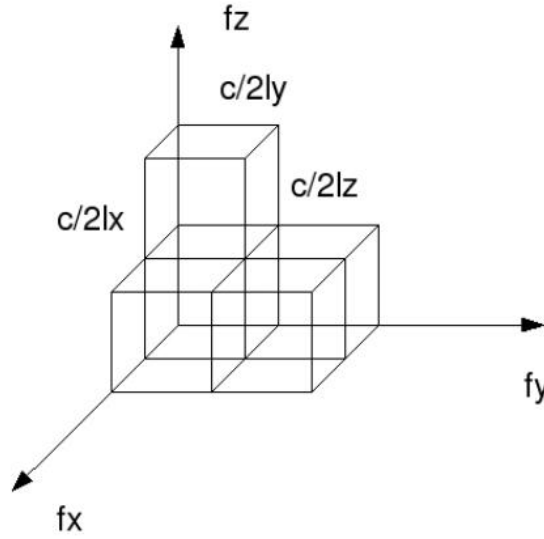


Figura 2.7: Cameră paralelipipedică

Din reprezentarea grafică se poate determina numărul (N) al frecvențelor proprii cu valori mai mici decât o frecvență dată (f_*), ca fiind numărul de paralelipipe elementare al căror volum total este egal cu volumul unei optimi de sferă de rază f_* . De asemenea poate fi calculat numărul modurilor de vibrații axiale, tangențiale și oblice.

$$N_{ax} = \frac{f_x}{\frac{c}{2l_x}} + \frac{f_y}{\frac{c}{2l_y}} + \frac{f_z}{\frac{c}{2l_z}} = \frac{2f_*}{c}(l_x + l_y + l_z) = \frac{f_*L}{2c}$$

$$N_{tg} = \frac{\frac{\pi f_*^2}{4}}{\frac{c}{2l_x} \frac{c}{2l_y}} + \frac{\frac{\pi f_*^2}{4}}{\frac{c}{2l_y} \frac{c}{2l_z}} + \frac{\frac{\pi f_*^2}{4}}{\frac{c}{2l_z} \frac{c}{2l_x}} - N_{ax} = \frac{\pi f_*^2 A}{2c^2} - \frac{f_*L}{2c}$$

$$N_{obl} = N - \frac{1}{2}N_{tg} - \frac{1}{4}N_{ax} = \frac{4\pi f_*^3 V}{3c^3} - \frac{\pi f_*^2 A}{4c^2} + \frac{f_* L}{8c}$$

$$L = 4(l_x + l_y + l_z) \quad \text{perimetrul paralelipipedului}$$

$$A = 2(l_x l_y + l_y l_z + l_z l_x) \quad \text{aria laterala a paralelipipedului}$$

$$V = l_x l_y l_z \quad \text{volumul paralelipipedului}$$

N este numărul total de frecvențe:

$$N = \frac{4\pi f_*^3 V}{3c^3} + \frac{\pi f_*^2 A}{4c^2} + \frac{f_* L}{8c}$$

2.2 Metode numerice de rezolvare

Pentru rezolvarea problemelor de acustică, înainte de apariția de fizicii computaționale, metodele de rezolvare preponderent folosite erau metodele grafice, care aveau avantajul că nu implicau un efort numeric deosebit de mare. Volumul de calcul necesar rezolvării folosind metode numerice, fiind prea mare, a descurajat folosirea acestui tip de metode până la dezvoltarea de sisteme de calcul puternice, capabile de prelua efortul computațional. Pe lângă acest lucru, de cele mai multe ori problemele de calcul în regim staționar devin deosebit de complexe, mai ales din prisma formelor geometrice folosite, sau a condițiilor la limită care variază în timp. Din această cauză, în ziua de azi, problemele de simulare se rezolvă utilizând metode numerice implementate în algoritmi computaționali puternic specializați și personalizați pentru fiecare problemă în parte.

Pentru aproximarea soluțiilor ecuației diferențiale a propagării undelor, putem folosi diverse metode numerice, dintre acestea: MDF (Metoda Diferențelor Finite), MEF (Metoda Elementelor Finite), MVF (Metoda Volumelor Finite). Fiecare dintre acestea produce un anumit tip de discretizare, în funcție de modul de rezolvare și de schema folosită.

În prezenta lucrare, pentru algoritmul de simulare a fost implementată o metoda diferențelor finite, folosind schema explicită de rezolvare. Discretizarea aleasă reprezintă un domeniu dreptunghiular de puncte (noduri) cu coordonate carteziene ortogonale. Pentru această metodă numerică, factorul principal care determină exactitatea soluției este granularitatea nodurilor implicate în calcul. Folosind un grid din ce în ce mai fin, soluția oferită de program va crește în acuratețea simulării oferite.

Având în vedere discretizarea, precum și faptul că pentru a simula propagarea unei unde acustice prin medii omogene considerăm vâscozitatea egală cu zero, vom stabili pentru domeniul dreptunghiular valorile presiunilor ca fiind necunoscutele aflate în noduri. Vom nota nodurile structurii de tip grid cu n . Presupunând o secțiune plană a spațiului pentru care se realizează simularea în regim nestaționar, propagarea unei acustice prin acesta se face după direcțiile O_x și O_y . Fiecare nod n are coordonate (i, j) , unde i reprezintă poziția pe axa O_x iar j poziția pe axa O_y . Considerând distanța între două noduri pe axa O_x ca fiind Δx , iar distanța pe axa O_y ca fiind Δy , adică pașii de discretizare, ecuația diferențială a propagării unde acustice, în regim nestaționar este:

$$\frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} + \frac{\partial^2 p}{\partial z^2} = \frac{1}{c^2} \cdot \frac{\partial^2 p}{\partial t^2}$$

Astfel, vom obține discretizarea bidimensională din figura (6):

Prin urmare nodurile vor fi numerotate astfel: $n_{i-1,j-1}$, $n_{i-1,j}$, $n_{i-1,j+1}$, $n_{i,j-1}$, $n_{i,j}$, etc. Folosind aceste notații precum și ecuația (63), vom scrie gradientii de presiune:

$$\begin{aligned} \frac{\partial p}{\partial x} \Big|_{i+\frac{1}{2},n} &\cong \frac{t_{i+1,j} - t_{i,j}}{\Delta x} \\ \frac{\partial p}{\partial y} \Big|_{i,j+\frac{1}{2}} &\cong \frac{t_{i,j+1} - t_{i,j}}{\Delta y} \\ \frac{\partial p}{\partial x} \Big|_{i-\frac{1}{2},j} &\cong \frac{t_{i,j} - t_{i-1,j}}{\Delta x} \end{aligned}$$

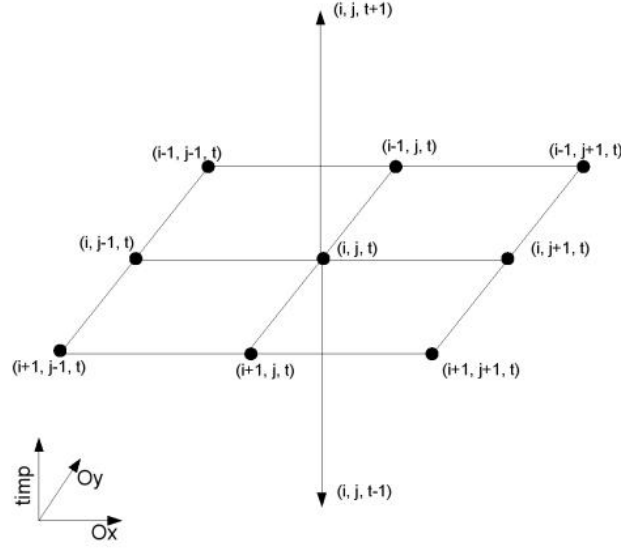


Figura 2.8: Discretizarea spațiului

$$\frac{\partial p}{\partial y}|_{i,j-\frac{1}{2}} \cong \frac{t_{i,j} - t_{i,j-1}}{\Delta y}$$

În aceeași idee obținem:

$$\frac{\partial^2 p}{\partial t^2} = \frac{\partial p}{\partial t} \cdot \left(\frac{p_{i,j}^{t+1} - p_{i,j}^t}{\Delta x} \right) = \frac{\frac{\partial p}{\partial t}(p_{i,j}^{t+1}) - \frac{\partial p}{\partial t}(p_{i,j}^t)}{\Delta t} = \frac{1}{\Delta t} \cdot \frac{u_{i,j}^{t+1} - u_{i,j}^t}{\Delta t} - \frac{1}{\Delta t} \cdot \frac{u_{i,j}^t - u_{i,j}^{t-1}}{\Delta t}$$

Deci:

$$\frac{\partial^2 p}{\partial t^2} = \frac{u_{i,j}^{t+1} - 2u_{i,j}^t + u_{i,j}^{t-1}}{\Delta t^2} \quad (2.64)$$

Folosind ecuația (64) împreună cu notațiile pentru gradientii de presiune, putem rescrie ecuația undelor acustice astfel:

$$\frac{u_{i,j}^{t+1} - 2u_{i,j}^t + u_{i,j}^{t-1}}{\Delta t^2} = \frac{u_{i+1,j}^t - 2u_{i,j}^t + u_{i-1,j}^t + u_{i,j+1}^t - 2u_{i,j}^t + u_{i,j-1}^t}{\Delta x^2} \quad (2.65)$$

Ecuația (65) reprezintă aproximarea numerică pentru ecuația diferențială a propagării acustice, care se poate rezolva, așa cum este cazul și în lucrarea de față, folosind schema explicită de calcul. Aceasta oferă acuratețe sporită când vine vorba de discretizarea temporală a procesului de propagare față de schema implicită, deoarece calculează starea domeniului la mai mulți pași între două intervale de timp oarecare, însă pe de altă parte, tocmai această granularitate temporală crescută

duce la performanțe mai scăzute față de schema implicită atunci când timpul fizic simulat depășește un anumit prag.

Deoarece avem de a face cu o metodă numerică, trebuie avut în vedere și aspectul stabilității calculelor. Pentru a face această analiză vom pleca de la formula:

$$u_{i,j}^{n+1} - 2u_{i,j}^n + u_{i,j}^{n-1} = \frac{(\Delta t)^2}{(\Delta x)^2} (u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n + u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n) \quad (2.66)$$

Rezultă:

$$u_{i,j}^{n+1} = 2u_{i,j}^n - u_{i,j}^{n-1} + \lambda (u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n + u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n) \quad (2.67)$$

Am notat cu λ valoarea raportului $\frac{(\Delta t)^2}{(\Delta x)^2}$. Dacă se notează cu ε^k eroarea introdusă la momentul k de timp se poate scrie relația:

$$\varepsilon^{n+1} = 2\varepsilon^n - \varepsilon^{n-1} + \varepsilon^n \lambda 4(c-1) \quad (2.68)$$

Se consideră că raportul erorilor introduse la pașii anteriori $\left(\frac{\varepsilon^{n-1}}{\varepsilon^n}\right)$ este egal cu 1 și se calculează valoarea raportului erorii introduse la pasul curent.

$$\frac{\varepsilon^{n+1}}{\varepsilon^n} = 2 - \frac{\varepsilon^{n-1}}{\varepsilon^n} + 4\lambda(c-1) \quad (2.69)$$

Notăm cu $G = \frac{\varepsilon^{n+1}}{\varepsilon^n}$

$$G = 2 - 1 + 4\lambda(c-1) = 1 + 4\lambda(c-1) \quad (2.70)$$

Impunem $G^2 \leq 1$ $(1 + 4\lambda(c-1))^2 \leq 1$ Rezultă:

$$1 + 8\lambda(c-1) + 16\lambda^2(c-1)^2 \leq 1$$

$$\Delta = (c-1)^2$$

$$\lambda_1 = 0 \quad \lambda_2 = -\frac{1}{2(c-1)}$$

Rezultă astfel condiția de stabilitate pentru metoda numerică utilizată:

$$\frac{(\Delta t)^2}{(\Delta x)^2} \leq \frac{1}{4} \quad (2.71)$$

Capitolul 3

Implementare

3.1 Inițializarea parametrilor de rulare

Întrucât aplicația se dorește a fi parte dintr-un pachet de aplicații de simulare CFD, o atenție deosebită a fost acordată modului de citire și inițializare a datelor de intrare. De aceea, pentru a oferi utilizatorului posibilitatea de a fi obligat să interacționeze cu aplicația cât mai puțin în timpul rulării a fost dezvoltată ideea de a rula multiple scenarii de simulare configurate în același fișier de intrare. Astfel, utilizatorul precizează numărul de scenarii pe care dorește să le ruleze, apoi, folosind identificatori de la 0 la numărul de scenarii dat, se va descrie fiecare scenariu în parte cu parametrii săi.

De asemenea, utilizatorul are și posibilitatea de a specifica în fișierul de configurare un număr de structuri de tip dreptunghic date prin pozițiile relative ale punctelor din colțuri față de marginile domeniului și care au proprietăți acustice reflexive, precum și poziția (tot relativ la marginile domeniului), raza și valoarea maxima de emisie a sursei. Mai jos este prezentat un exemplu de fișier de configurare a parametrilor de rulare:

```
[NUM_SCENARIOS] = 1
```

```
[SCENARIO] = 0
```

```
[SIZEX] = 150
```

```
[SIZEY] = 150
```

[H] = 0.25
[MAX_TIME] = 30
[TIME_STEP] = 0.1
[SAVE_TIME] = 30
[SRC_X] = 0.2
[SRC_Y] = 0.4
[SRC_RADIUS] = 4
[SRC_AMPLITUDE] = 200
[NUM_STRUCTURES] = 5
[STRUCTURE] = 0
[P1] = 0.5 0.2
[P2] = 0.5 0.37
[P3] = 0.6 0.37
[P4] = 0.6 0.2
[STRUCTURE] = 1
[P1] = 0.5 0.43
[P2] = 0.5 0.6
[P3] = 0.6 0.43
[P4] = 0.6 0.6
[STRUCTURE] = 2
[P1] = 0.1 0.2
[P2] = 0.1 0.22
[P3] = 0.25 0.22
[P4] = 0.25 0.2
[STRUCTURE] = 3
[P1] = 0.1 0.58
[P2] = 0.1 0.6
[P3] = 0.25 0.6
[P4] = 0.25 0.58
[STRUCTURE] = 4
[P1] = 0.1 0.25
[P2] = 0.1 0.55
[P3] = 0.12 0.55
[P4] = 0.12 0.25

[SCENARIO] = 1
[SIZE_X] = 150
[SIZE_Y] = 150
[H] = 1.2
[MAX_TIME] = 45
[TIME_STEP] = 0.5
[SAVE_TIME] = 3
[SRC_X] = 0.2
[SRC_Y] = 0.4
[SRC_RADIUS] = 4
[SRC_AMPLITUDE] = 125
[NUM_STRUCTURES] = 0

3.2 Algoritmul serial

Pentru implementarea metodei cu diferențe finite și schemă implicită va trebui să tratăm în cod discretizarea domeniului fizic pentru care se face simularea folosind matrice de elemente cu valori în virgulă mobilă. Pentru cazul de față am considerat că folosirea tipului de date "double" este suficientă pentru a exprima calculele în limitele de aproximație dorite. Dacă vom considera forma folosită pentru a rezolva ecuația propagării undelor constatăm că vom avea nevoie de 3 matrice de numere în virgulă mobilă, fiecare reprezentând un nod a cărui necunoscută este valoarea presiunii. O abordare echivalentă ar fi fost transpunerea discretizării în termeni computaționali în liste înlanțuite de noduri, unde pentru fiecare patru noduri se definea o valoare reprezentând presiunea aproximativă din centrul celulei determinate de cele patru noduri.

Algoritmul iterează calculele asupra matricelor timp de un număr de iterații egal cu numărul de subdiviziuni de lungime Δt care descriu durata fizică reală a procesului. Acești doi parametri (*MAX_TIME* și *TIME_STEP*) sunt preluați din fișierul de intrare. Deoarece este vorba de simularea unui proces în regim staționar nu avem alte condiții de oprire (cum ar fi convergența în cazul simulărilor pentru procese în regim staționar) de aceea fiind neapărat necesară impunerea condițiilor de stabilitate pentru asigurarea corectitudinii algoritmului.

Deoarece fișierul de intrare permite definirea unor structuri dreptunghiulare cu proprietăți acustice reflexive, algoritmul are ca primă sarcină detectarea tipului de mediu pe care fiecare celulă (sau nod) în parte îl simulează. De asemenea, se ia în considerare și faptul că celulele care constituie muchia unei structuri sau marginile domeniului se tratează diferit de celulele din colțurile unei structuri sau colțurile domeniului discretizat. Fiecare dintre aceste tipuri de noduri menționate reflectă unda pe o direcție dată, ceea ce traducem în termeni numerici ca valoarea nodului de pe o muchie să fie egală cu valoarea nodului vecin care se afla în domeniu (pentru granițele domeniului) sau egală cu valoarea nodului vecin care nu se află în interiorul

structurii (pentru muchia unei structuri). Din această cauză, au fost definite o serie de funcții de dimensiuni mici care testează sau calculează valoarea unui nod, după caz.

Funcție care testează dacă o celulă face parte din sursă:

```
int is_source(int x, int y, int radius, int source_active)
{
    if(!source_active)
        return 0;
    if(sqrt(pow(scenario[scn_index].source.x-x,2)+pow(scenario[scn_index].source.y-y,2))
        <= radius)
        return 1;
    return 0;
}
```

Funcție care testează dacă o celulă face parte din muchia domeniului:

```
int on_edge(int x, int y)
{
    if(x == 0 && y != 0 && y != nx-1)
        return N_EDGE;
    if(x == ny-1 && y != 0 && y != nx-1)
        return S_EDGE;
    if(y == 0 && x != 0 && x != ny-1)
        return W_EDGE;
    if(y == nx-1 && x != 0 && x != ny-1)
        return E_EDGE;
    return 0;
}
```

Funcție care realizează calculul pentru o celulă aflată în colțul unei structuri:

```
double compute_structure_corner_node(int i, int j, int corner)
{
    switch(corner)
    {
        case NW_CORNER:
            return (ub[i][j-1]+ub[i-1][j])/2;
        case NE_CORNER:
            return (ub[i-1][j]+ub[i][j+1])/2;
        case SE_CORNER:
            return (ub[i][j+1]+ub[i+1][j])/2;
        case SW_CORNER:
            return (ub[i+1][j]+ub[i][j-1])/2;
        default:
            return 0;
    }
}
```

Tot din fișierul de intrare se setează un alt parametru (numit *SAVE_TIME*) care specifică după câte iterații de timp se efectuează salvarea în fișier a stării curente a domeniului. Salvarea se va face folosind datele din matrice pentru crearea unui fișier de tip VTK, în care definim un grid nestructurat, unde fiecare celulă este mărginită de 4 noduri, iar centrul celulei reprezintă valoarea aproximată a presiunii pentru elementul spațial considerat.

3.3 Paralelizare cu OpenMP

OpenMP este cel mai folosit API pentru crearea de aplicații paralele cu memorie partajată în limbaje precum C, C++ și Fortran. Implementarea unei aplicații folosind OpenMP presupune lucrul cu directive de compilator, funcții (rutine de rutare) și variabile de mediu. În general, librăria este larg folosită pentru a paraleliza buclele de calcul care necesită efort computațional mare. Totuși, trebuie avut în vedere că folosirea directivelor de paralelizare nu garantează speed-up decât dacă se îndeplinesc câteva condiții, cum ar fi: determinarea apriori de către programator a gradului de paralelizare a unei bucăți de cod, sau asigurarea unei granularități sporite calculelor pentru a oferi posibilitatea compilatorului de a paraleliza cât mai mult posibil.

Paralelizarea cu OpenMP este de fapt, un proces transparent pentru programator. Pentru o regiune de cod paralelizată cu OpenMP se crează mai multe thread-uri care execută regiunea de cod simultan, cu unul dintre thread-uri în rolul de "master", iar celelalte în rolul de "workers", toate formând un "OpenMP Team". Înainte de lansarea în execuție a thread-urilor, folosind un mecanism specific, OpenMP distribuie sarcinile pentru fiecare thread în parte. De asemenea, ne sunt puse la dispoziție instrumente, sub forma unor clauze pentru directivele OpenMP, prin care putem face teste, sau putem aloca variabile private pentru fiecare thread în parte, ori variabile partajate de toate thread-urile.

Mai jos sunt câteva exemple de utilizări în cadrul codului a directivelor OpenMP. Pentru bucla care anulează sursa după ce timpul de emisie alocat acesteia expiră:

```
#pragma omp parallel for private(i,j)
for(i=0;i<ny;i++)
for(j=0;j<nx;j++)
{
    if(is_source(i,j,radius,source_active))
        uc[i][j] = ub[i][j] = ua[i][j] = 0;
}
```

Pentru bucla principală de calcul existența numărului crescut de instrucțiuni *if-else* sporește efortul computațional pentru rezolvarea serială, dar efectele acestei probleme scad simțitor prin folosirea OpenMP:

```
#pragma omp parallel for private(i,j)
for(i=0;i<ny;i++)
for(j=0;j<nx;j++)
{
    if(!on_corner(i,j) && !on_edge(i,j) && !is_source(i,j,radius,source_active) &&
        && !on_structure_edge(i,j) && !on_structure_corner(i,j) && !in_structure(i,j))
        uc[i][j] = compute_node(i,j);
    else if((place = on_edge(i,j)))
        uc[i][j] = compute_edge_node(i,j,place);
    else if((place = on_corner(i,j)))
        uc[i][j] = compute_corner_node(i,j,place);
    else if((place = on_structure_edge(i,j)))
        uc[i][j] = compute_structure_edge_node(i,j,place);
    else if((place = on_structure_corner(i,j)))
        uc[i][j] = compute_structure_corner_node(i,j,place);

    ua[i][j] = 0;
}
```

Avantajele OpenMP sunt numeroase: scalează foarte bine (mai ales pe sistemele multicore), are performanțe bune și scalabilitate mare în general, este extrem de portabil oferind o listă largă de compatibilitate cu diverse compilatoare, nu necesită prea mult efort de programare și permite paralelizarea incrementală a aplicațiilor.

3.4 Paralelizare și calcul distribuit cu MPI

MPI (Message Passing Interface) este o specificație pentru un API care permite mai multor sisteme de calcul să comunice între ele. Este folosit îndeosebi în apli-

cațiile care rulează peste clustere sau pe supercalculatoare. MPI este în fapt un protocol de comunicație independent de limbaj, care suporta comunicații atât punct-la-punct, cât și broadcast. Cu toate că nu a fost standardizat de nicio instituție de competență în domeniu, MPI a devenit standardul *de-facto* pentru implementarea aplicațiilor ce modelează un proces paralelizat care rulează pe un sistem cu memorie distribuită, deși poate rula fără probleme (și cu performanțe destul de bune) și pe sisteme cu memorie partajată.

Majoritatea implementărilor MPI constau într-un set de rutine (constituite într-un API) apelabile din aplicații scrise în limbajele C, C++ sau Fortran. Interfața MPI are ca scop crearea unei topologii virtuale de procese, cărora le oferă instrumente de sincronizare și funcționalități de comunicație inter-procese. În mod normal, pentru a obține performanțe maxime, fiecare proces este în relație de corespondență unu-la-unu cu un procesor din cadrul sistemului, această distribuție și mapare a proceselor peste resursele hardware făcându-se la rulare de către un agent specializat (de obicei *mpirun* sau *mpiexec*).

Conceptele care stau la baza MPI-ului sunt:

- **Comunicatorul**

Entitate care interconectează și grupează procesele aflate în rulare. Fiecare proces din cadrul comunicatorului primește un identificator (numit *rank*) generându-se odată cu sesiunea MPI (care pornește prin apelarea funcției *MPI_Init()*) o topologie ordonată de procese.

- **Instrumente punct-la-punct**

Un număr mare de funcții importante au fost implementate pentru comunicațiile între două procese, dintre care cele mai folosite sunt *MPI_Send()* și *MPI_Recv()*. Acest tip de comunicație, datorită naturii sale ușor adaptabile la comunicații repetate care formează un șablon, se constituie într-o unealtă extrem de utilă pentru aplicația implementată în această lucrare.

- **Instrumente de comunicație colectivă**

Acest tip de funcții oferă proceselor capacitatea de a comunica cu toate procesele din grup sau cu un subset al acestora. De exemplu, funcția *MPI_Bcast()* trimite date aparținând unui proces către toate celelalte procese din grup, sau funcția *MPI_Reduce()* care preia date din toate procesele unui grup, efectuează o operație asupra lor (specificată de programator) și apoi salvează rezultatul obținut pe un anumit nod.

- **Tipuri de date derivate**

Deoarece MPI lucrează cu funcții care necesită pasarea de argumente care specifică tipul variabilelor, uneori programatorul este nevoit să definească noi tipuri de date pentru MPI în ideea de a ușura și eficientiza transmisiile de date. Acestea se creează folosind funcțiile *MPI_Type_struct()*, *MPI_Type_commit()*, *MPI_extent()*, *MPI_Get_address()*, etc.

În implementarea aplicației, nu au fost definite noi comunicatoare, preferându-se cel default deoarece rularea și comunicația proceselor la testare au avut loc într-un mediu omogen cu topologie simplă. Paradigma paralelă și distribuită aleasă este "boss-worker" deoarece permite centralizarea ușoară a datelor și salvarea rezultatelor parțiale într-un mod coerent și eficient.

Având în vedere natura algoritmului implementat (diferențe finite cu schemă implicită centrată), în faza de proiectare a aplicației s-a făcut remarcată nevoia unui sistem de interschimbare de date inter-procese la fiecare iterație, cu posibilitatea de a "agrega" datele pentru salvarea în fișier. Soluția găsită se bazează pe identificatorul proceselor și constă în trimiterea mai întâi a datelor de la procesele cu identificator impar către cele cu identificator par, apoi trimiterea datelor de la procesele cu identificator la par la cele cu identificator impar. De remarcat însă că pentru a realiza interschimbarea datelor între procese a fost necesară alocarea de spațiu suplimentar în fiecare proces în parte pentru a putea acomoda și mai apoi folosi direct în bucla de calcul datele pe care fiecare proces le primește.

Mai jos este prezentată soluția implementată:

```
if(rank == 0)
{
    MPI_Recv(uc[local_ny],nx,MPI_DOUBLE,1,1,MPI_COMM_WORLD,&status);
    MPI_Send(uc[local_ny-1],nx,MPI_DOUBLE,1,1,MPI_COMM_WORLD);
}
else if(rank%2 == 1 && rank != numtask-1)
{
    MPI_Send(uc[1],nx,MPI_DOUBLE,rank-1,1,MPI_COMM_WORLD);
    MPI_Recv(uc[0],nx,MPI_DOUBLE,rank-1,1,MPI_COMM_WORLD,&status);
    MPI_Send(uc[local_ny],nx,MPI_DOUBLE,rank+1,1,MPI_COMM_WORLD);
    MPI_Recv(uc[local_ny+1],nx,MPI_DOUBLE,rank+1,1,MPI_COMM_WORLD,&status);
}
else if(rank%2 == 0 && rank != numtask-1)
{
    MPI_Recv(uc[local_ny+1],nx,MPI_DOUBLE,rank+1,1,MPI_COMM_WORLD,&status);
    MPI_Send(uc[local_ny],nx,MPI_DOUBLE,rank+1,1,MPI_COMM_WORLD);
    MPI_Recv(uc[0],nx,MPI_DOUBLE,rank-1,1,MPI_COMM_WORLD,&status);
    MPI_Send(uc[1],nx,MPI_DOUBLE,rank-1,1,MPI_COMM_WORLD);
}
else if(rank == numtask-1)
{
    MPI_Send(uc[1],nx,MPI_DOUBLE,rank-1,1,MPI_COMM_WORLD);
    MPI_Recv(uc[0],nx,MPI_DOUBLE,rank-1,1,MPI_COMM_WORLD,&status);
}
```

Pentru a optimiza inițializarea datelor înainte de rulare și având în vedere ca citirea datelor se face centralizat (de către rank-ul "boss") a fost necesară definirea câtorva tipuri de date, structuri, pentru comunicația între rank-ul "boss" și rank-urile "worker". Soluția aleasă a fost astfel implementată:

```
str_offsets[0] = 0;
str_types[0] = MPI_INT;
str_blockcounts[0] = 8;

MPI_Type_struct(1,str_blockcounts,str_offsets,str_types,&MPI_STRUCTURE);
MPI_Type_commit(&MPI_STRUCTURE);

src_offsets[0] = 0;
src_types[0] = MPI_INT;
src_blockcounts[0] = 3;

MPI_Type_struct(1,src_blockcounts,src_offsets,src_types,&MPI_SOURCE);
MPI_Type_commit(&MPI_SOURCE);

scn_offsets[0] = 0;
scn_types[0] = MPI_INT;
scn_blockcounts[0] = 4;

MPI_Type_extent(MPI_INT,&extent);
scn_offsets[1] = scn_offsets[0] + scn_blockcounts[0]*extent;
```

```

scn_types[1] = MPI_DOUBLE;
scn_blockcounts[1] = 4;

MPI_Type_extent(MPI_DOUBLE,&extent);
scn_offsets[2] = scn_offsets[1] + scn_blockcounts[1]*extent;
scn_types[2] = MPI_SOURCE;
scn_blockcounts[2] = 1;

MPI_Type_extent(MPI_SOURCE,&extent);
scn_offsets[3] = scn_offsets[2] + scn_blockcounts[2]*extent;
scn_types[3] = MPI_STRUCTURE;
scn_blockcounts[3] = MAX_STRUCTURES;

MPI_Type_struct(4,scn_blockcounts,scn_offsets,scn_types,&MPI_SCENARIO);
MPI_Type_commit(&MPI_SCENARIO);

```

Făcând uz de avantajele implementării cu OpenMP, în implementarea folosind MPI a fost integrată, cu destul de multă ușurință, și paralelizarea regiunilor critice de calcul folosind OpenMP. Acest lucru sporește și mai mult performanțele aplicației, creând o implementare hibridă care profită atât de capacitățile calculului distribuit ale MPI-ului, cât și de cele de calcul paralel folosind memorie partajată a OpenMP-ului foarte utilă mai ales când nodul alocat este multicore.

3.5 Salvarea datelor în fișier

Salvarea datelor în fișier este făcută numai de procesul "boss" care centralizează datele de execuție la fiecare *SAVE_STEP* iterații și le scrie în fișier. Fișierele sunt special create cu denumiri care să sugereze cărui scenariu fac parte și cărui pas din simulare îi corespunde, acest lucru ușurând și vizualizarea deoarece acestea vor fi grupate după nume. Export-ul datelor în format VTK face ca rezultatele, chiar dacă au dimensiuni mari, să poată fi ușor vizualizate cu programe specializate cum este și Paraview (care poate face randări de vizualizare în mod paralel).

Capitolul 4

Testare

4.1 Arhitectura sistemului de testare

Clusterul avut la dispoziție pentru testare are următoarea structură:

Număr Unități	Tip
51	Pentium 4 Hyper Threading
32	Dual Xeon
32	Dual-Quad Core Xeon
4	Dual PowerXCell8i

Figura 4.1: Structura clusterului

Toate cele 3 implementări au fost testate folosind, din cadrul cluterului prezentat mai sus, 4 sisteme Dual-Quad Core Xenon, totalizând 16 core-uri de procesare.

Clusterul folosește Sun Grid Engine, un batch-queuing system open-source dezvoltat de către cei de la Sun. SGE, cum mai este cunoscut, are rolul de a prelua, planifica, distribui și superviza execuția remote și distribuită a unui număr mare de job-uri unice, paralele și/sau interactive. De asemenea, supervizează și planifică alocarea și distribuția de resurse precum procesoarele, memoria, spațiul pe disc și licențele software.

O configurație tipică pentru un cluster administrat cu Sun Grid Engine constă într-un nod master și mai multe noduri de execuție. De asemenea, există posibilitatea configurării unor noduri master "de rezervă", care pot prelua imediat sarcinile master-ului original în cazul în care acesta devine neoperațional.

Pentru a trimite un "job" spre execuția pe cluster se folosește comanda *qsub* care primește anumiți parametri. Câteva dintre aceștia sunt:

- **-q**

Specifică coada de job-uri spre care utilizatorul vrea să trimită job-ul pentru rulare. Sunt de obicei folosite pentru a grupa într-un "pool" comun toate resursele de același tip, cu scopul "omogenizării".

- **-pe nume număr**

Specifică un mediu paralel de rulare care permite execuția de aplicații paralelizate care folosesc memorie partajată sau distribuită. Un exemplu de acest mediu este și cel folosit pentru MPI. Argumentul *n* specifică numărul de unități de procesare dorite a fi alocate pentru execuție.

- **-cwd path**

Setează directorul curent ca fiind directorul de lucru "virtual" pentru aplicația ce va fi rulată. Astfel toate fișierele rezultate în urma rulării sau necesare rulării vor fi accesate via NFS direct din "path".

Pentru trimiterea unui job pe cluster prin intermediul SGE s-a folosit comanda:

```
qsub -pe ibm-quad <număr_elemente_procesare> -cwd <script_rulare>
```

unde se precizează numărul de elemente de procesare necesare rulării precum și un script de rulare care are, de exemplu, forma (pentru o rulare în MPI):

```
#!/bin/bash
mpirun -np 4 ~/implementare/mpi/acoustics ~/implementare/mpi/input
```

4.2 Rezultate obținute

Pentru exemplificarea rulării s-a folosit următorul fișier de intrare pentru setarea parametrilor:

```
[NUM_SCENARIOS] = 1

[SCENARIO] = 0
[SIZEX] = 200
[SIZEY] = 200
[H] = 0.25
[MAX_TIME] = 30
[TIME_STEP] = 0.1
[SAVE_TIME] = 30
[SRC_X] = 0.3
[SRC_Y] = 0.4
[SRC_RADIUS] = 4
[SRC_AMPLITUDE] = 200
[NUM_STRUCTURES] = 5
[STRUCTURE] = 0
[P1] = 0.4 0.2
[P2] = 0.4 0.37
[P3] = 0.5 0.37
[P4] = 0.5 0.2
[STRUCTURE] = 1
[P1] = 0.4 0.43
[P2] = 0.4 0.6
[P3] = 0.5 0.6
[P4] = 0.5 0.43
[STRUCTURE] = 2
[P1] = 0.1 0.2
[P2] = 0.1 0.22
[P3] = 0.25 0.22
[P4] = 0.25 0.2
[STRUCTURE] = 3
[P1] = 0.1 0.58
[P2] = 0.1 0.6
[P3] = 0.25 0.6
[P4] = 0.25 0.58
[STRUCTURE] = 4
[P1] = 0.1 0.25
[P2] = 0.1 0.55
[P3] = 0.12 0.55
[P4] = 0.12 0.25
```

Se observă că este vorba de o rulare ce simulează 30 de secunde reale, cu pasul Δt de 0.1 secunde, pentru discretizarea folosindu-se 200 de celule pentru axa O_x și 200 de celule pentru axa O_y (40.000 de celule în total), cu pasul între celule de 0.25

unități, amplitudinea maximă a sursei de 200 de unități și raza sursei de 4 celule. De asemenea, scenariul include și 5 structuri dreptunghice.

Mai jos se găsește o secvență de rezultate ale simulării, în ordine temporală, pentru scenariul prezentat mai sus:

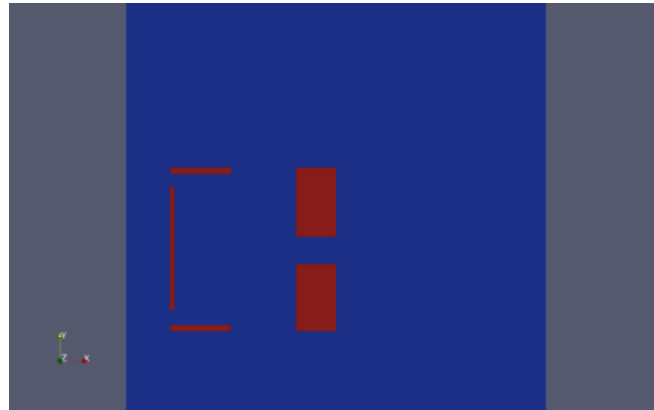


Figura 4.2: Simularea la secunda 0

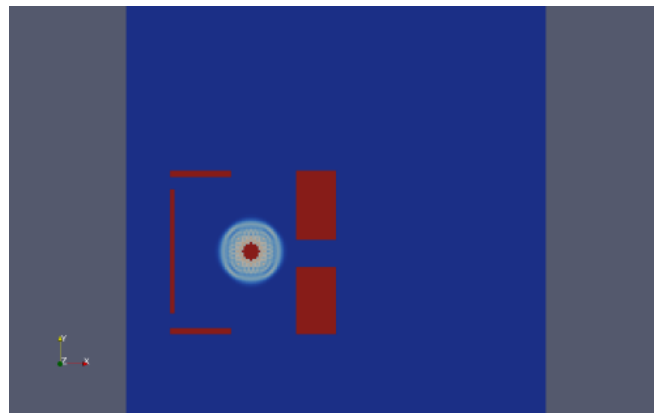


Figura 4.3: Simularea la secunda 6

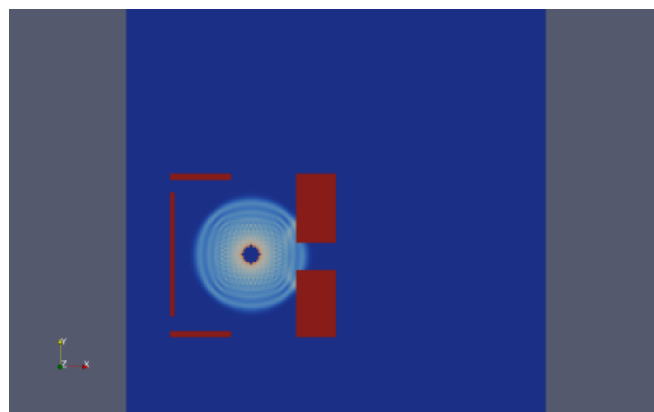


Figura 4.4: Simularea la secunda 9

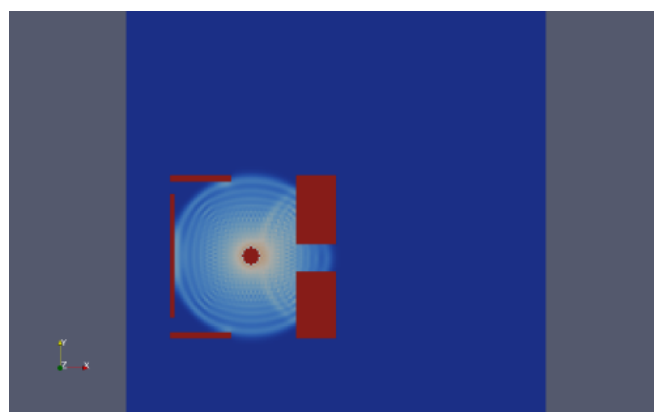


Figura 4.5: Simularea la secunda 12

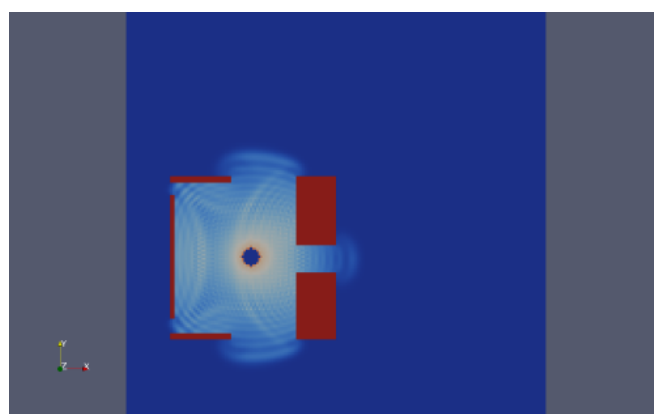


Figura 4.6: Simularea la secunda 15

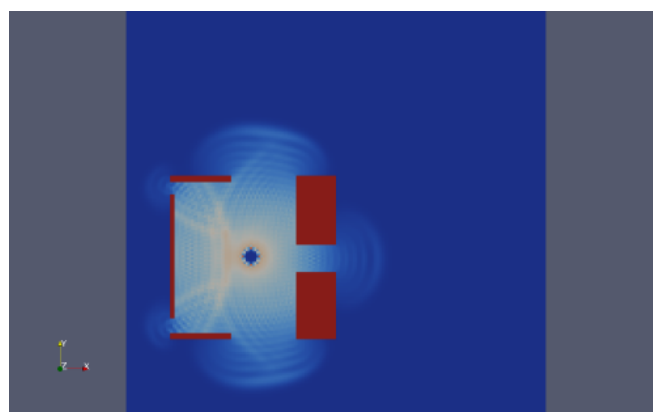


Figura 4.7: Simularea la secunda 18

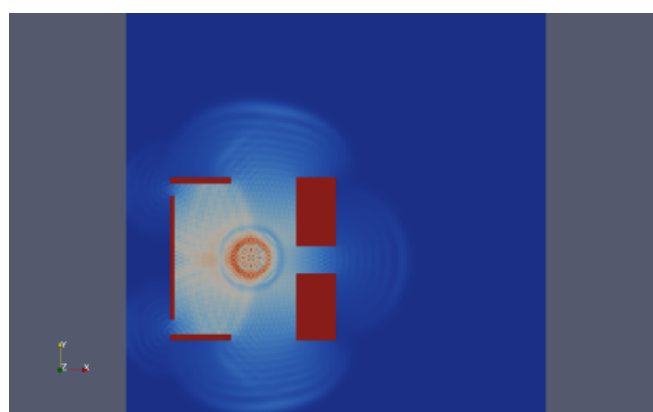


Figura 4.8: Simularea la secunda 21

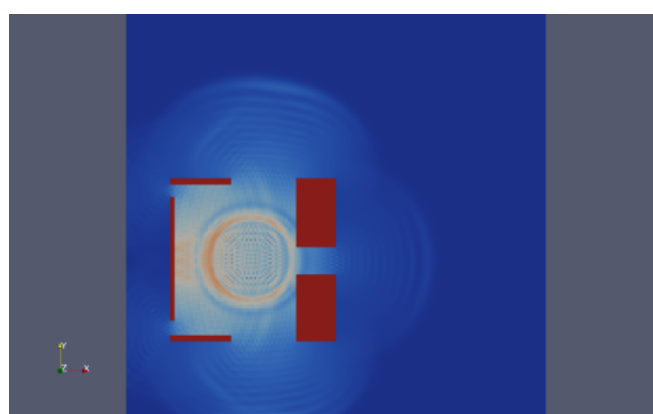


Figura 4.9: Simularea la secunda 24

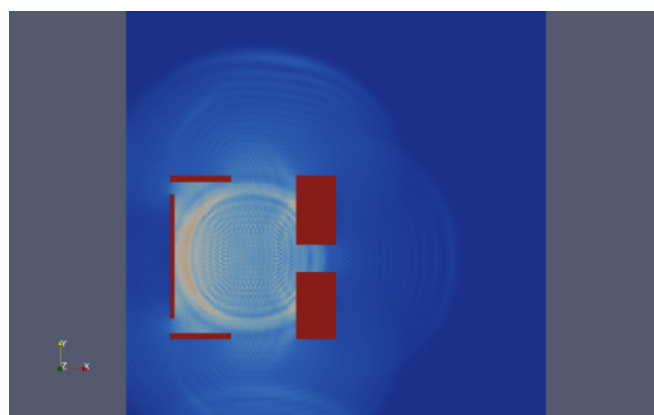


Figura 4.10: Simularea la secunda 27

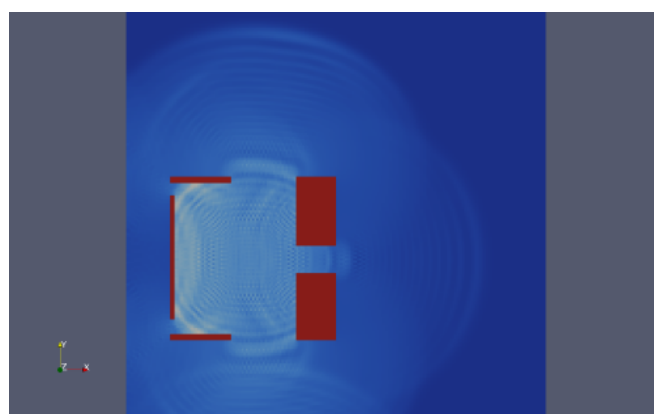


Figura 4.11: Simularea la secunda 30

Capitolul 5

Performanțe

5.1 Profiling

În timpul procesului de dezvoltare a aplicației, în cadrul etapei de optimizare, am folosit Sun Studio Analyzer, un instrument de analiză a performanțelor care permite descoperirea regiunilor de cod care afectează performanțele. Această aplicație este constituită, de fapt din două instrumente. Primul, colectorul, este cel care urmărește și contorizează apelurile de funcții din program, librării sau apelurile de sistem. Cel de-al doilea, analizorul, permite vizualizarea datelor colectate despre aplicația testată și prezintă, folosind diferite metrici, performanțele programului. Metricile pot fi: de timp, de contoare hardware, de întârzieri datorită sincronizărilor, de alocare de memorie sau de MPI.

În continuare se va prezenta modalitatea prin care s-a efectuat una dintr optimizările din cod. În timpul rulării unui scenariu de dimensiunea de 40.000 de celule și 600 de iterații temporale, datele au fost colectate într-un "experiment".

După terminare, la interpretarea datelor se afișă:

Functions		Callers-Callees		Source	Disassembly	Timeline	Experiments
User CPU		User CPU		Name			
▼ (sec.)	(%)	(sec.)	(%)				
16.189	100.00	16.189	100.00	<Total>			
9.733	60.12	16.134	99.66	<libc-2.5.so>			
2.980	18.41	2.980	18.41	<libm-2.5.so>			
0.715	4.42	0.715	4.42	on_structure_edge			
0.638	3.94	0.638	3.94	on_structure_corner			
0.539	3.33	1.969	12.16	compute_node			
0.506	3.12	0.506	3.12	in_structure			
0.473	2.92	2.024	12.50	is_source			
0.264	1.63	16.134	99.66	s_compute_acoustics			
0.121	0.75	0.121	0.75	on_edge			
0.099	0.61	0.099	0.61	<libpthread-2.5.so>			
0.066	0.41	9.920	61.28	export_to_vtk			
0.044	0.27	0.044	0.27	on_corner			
0.011	0.07	0.990	6.11	pulse_source			
0.	0.	16.134	99.66	main			

Figura 5.1: Exemplu de experiment (main)

Evident, funcția care trezește interes este *s_compute_acoustics*, cea care ocupa cel mai mare timp de rulare:





Functions		Callers-Callees		Source		Disassembly		Timeline		Experiments	
 User CPU		 User CPU		 User CPU				Name			
▼ (sec.)	(%)	(sec.)	(%)	(sec.)	(%)						
16.134	100.00	0.	0.	16.134	99.66	main					
											
0.264	1.64	0.264	1.63	16.134	99.66	s_compute_acoustics					
9.920	61.49	0.066	0.41	9.920	61.28	export_to_vtk					
1.958	12.13	0.539	3.33	1.969	12.16	compute_node					
1.001	6.20	0.473	2.92	2.024	12.50	is_source					
0.990	6.13	0.011	0.07	0.990	6.11	pulse_source					
0.715	4.43	0.715	4.42	0.715	4.42	on_structure_edge					
0.638	3.95	0.638	3.94	0.638	3.94	on_structure_corner					
0.484	3.00	0.506	3.12	0.506	3.12	in_structure					
0.121	0.75	0.121	0.75	0.121	0.75	on_edge					
0.044	0.27	0.044	0.27	0.044	0.27	on_corner					

Figura 5.2: Exemplu de experiment (funcția *s_compute_acoustics*)

În funcția *s_compute_acoustics*, observăm că, pe lângă funcția care realizează salvarea datelor în fișier (*export_to_vtk*) care este mare consumatoare de resurse

având în vedere că este responsabilă cu salvarea pe disc a sute de MB, alte consumatoare de timp sunt și funcțiile *compute_node* și *is_source*. În cazul celei de-a doua funcții nu se mai pot face alte optimizări deoarece este o funcție de verificare ce consumă resurse datorită apelurilor "obligatorii" *sqrt* (necesare pentru păstrarea consistenței rezultatelor). În cazul primei funcții, codul înainte de optimizare avea forma:

```
double compute_node(int x, int y)
{
    return (2*ub[x][y] - ua[x][y] + (pow(TIME_STEP,2)/pow(H,2)) * (ub[x+1][y] -
        - 4*ub[x][y] + ub[x-1][y] + ub[x][y+1] + ub[x][y-1]));
}
```

Pentru această funcție, analizatorul indică resurse de timp de calcul consumate pentru apelul funcției *pow*:




Functions		Callers-Callees		Source		Disassembly		Timeline		Experiments	
 User	CPU	 User	CPU	 User	CPU			Name			
▼ (sec.)	(%)	(sec.)	(%)	(sec.)	(%)						
1.958	99.44	0.264	1.63	16.134	99.66			s_compute_acoustics			
0.011	0.56	16.189	100.00	16.189	100.00			<Total>			

Figura 5.3: Exemplu de experiment (functia *compute_node*)

Soluția de optimizare pentru această funcție este evidentă, în contextul în care *TIME_STEP* și *H* sunt nu se modifică pentru un scenariu dat. Astfel, a fost eliminată funcția *pow* și înlocuită cu o înmulțire, iar calculul respectiv a fost făcut la începutul scenariului și păstrat într-o variabilă globală. Funcția modificată arată astfel:

```
double compute_node(int x, int y)
{
    return (2*ub[x][y] - ua[x][y] + gain * (ub[x+1][y] -
        - 4*ub[x][y] + ub[x-1][y] + ub[x][y+1] + ub[x][y-1]));
}
```

După această schimbare a fost rulat un nou experiment, cu următorul rezultat:

Functions	Callers-Callees	Source	Disassembly	Timeline	Experiments
User CPU (sec.)	User CPU (sec.)	Name			
14.737	14.737	<Total>			
9.722	14.737	<libc-2.5.so>			
0.704	0.704	on_structure_edge			
0.682	0.682	on_structure_corner			
0.495	0.495	compute_node			
0.495	0.495	in_structure			
0.473	1.232	pow			
0.407	1.903	is_source			
0.385	14.737	s_compute_acoustics			
0.374	0.374	__ieee754_pow			
0.374	0.374	isnan			
0.121	0.264	sqrt			
0.110	0.110	<libpthread-2.5.so>			
0.099	0.099	__ieee754_sqrt			
0.077	9.953	export_to_vtk			
0.077	0.077	on_corner			
0.077	0.077	on_edge			
0.055	0.055	finite			
0.011	0.803	pulse_source			
0.	14.737	main			

Figura 5.4: Exemplu de experiment (funcția main)

Observăm că timpul de rulare total s-a redus, iar funcția de interes *s_compute_acoustics* a fost măsurată:

Functions	Callers-Callees	Source	Disassembly	Timeline	Experiments
User CPU (sec.)	User CPU (sec.)	User CPU (sec.)	Name		
14.737	0.	14.737	main		
0.385	0.385	14.737	s_compute_acoustics		
9.953	0.077	9.953	export_to_vtk		
1.111	0.407	1.903	is_source		
0.803	0.011	0.803	pulse_source		
0.704	0.704	0.704	on_structure_edge		
0.682	0.682	0.682	on_structure_corner		
0.495	0.495	0.495	compute_node		
0.429	0.495	0.495	in_structure		
0.077	0.077	0.077	on_corner		
0.077	0.077	0.077	on_edge		
0.022	9.722	14.737	<libc-2.5.so>		

Figura 5.5: Exemplu de experiment (funcția s_compute_acoustics)

Iar pentru funcția `compute_node` avem după optimizare măsurătorile:

Functions				Callers-Callees	Source	Disassembly	Timeline	Experiments
User CPU (sec.)	User CPU (sec.)	User CPU (sec.)	Name					
0.495	0.385	14.737	s_compute_acoustics					
0.495	0.495	0.495	compute_node					

Figura 5.6: Exemplu de experiment după optimizare (funcția `compute_node`)

În concluzie, prin rularea primului experiment, identificarea problemei de performanță, optimizarea reginuii de cod consumatoare de resurse de calcul și rularea celui de-al doilea experiment, observăm că pentru o modificare relativ minoră am obținut o scădere a duratei totale de rulare de aproximativ 12%, deci un speed-up de 1.14. Repetând ciclul descris mai sus s-au identificat și alte probleme care au fost rând pe rând rezolvate și care au îmbunătățit considerabil performanțele aplicației.

Un alt exemplu edificator, care scoate în evidență nu numai nevoia de a paraleliza calculul dar și necesitatea optimizării codului, este cel în care matricele care memorează cele trei stări temporale ale domeniului necesare calculului ($t+1$, t și $t-1$) sunt interschimbate după terminarea iterației de calcul (pentru refolosirea spațiului deja alocat în memorie). În versiunea inițială a codului, interschimbul se făcea copiind datele folosind două instrucțiuni de tip *for* imbricate. După analiza de optimizare, datele din cele trei matrice au fost interschimbate modificând doar pointerii către zonele de memorie alocate și ținând cont de faptul că matricea de la timpul $t-1$ devenea aceeași cu cea de la timpul t , iar matricea de la timpul t aceeași cu cea de la timpul $t+1$, în timp ce matricea de la timpul $t+1$ era oricum recalulată pentru fiecare celulă, deci putea adresa la iterația viitoare spațiul matricei la timpul $t-1$. O astfel de optimizare îmbunătățește substanțial performanța oricărui program care

lucrează cu matrice alocate dinamic care sunt modificate și interschimbate la fiecare iterație.

5.2 Timpi de rulare și speed-up

Pentru testarea performanțelor s-a folosit un fișier de configurare care conținea mai multe scenarii cu dimensiunea domeniilor de discretizare crescând între 40.000 de celule și 1.000.000 de celule. Numărul de iterații realizat pentru fiecare dintre aceste domenii este 600, număr ce reprezintă numărul cuantelor de timp obținute prin discretizarea temporală, totalizând 30 de secunde de simulare fizică.

Mai jos se regăsește scenariul folosit ca test pentru toate cele 3 tipuri de implementări:

```
[NUM_SCENARIOS] = 9

[SCENARIO] = 0
[OMP_THREADS] = 2
[SIZEX] = 200
[SIZY] = 200
[H] = 0.2
[MAX_TIME] = 30
[TIME_STEP] = 0.05
[SAVE_TIME] = 10
[SRC_X] = 0.3
[SRC_Y] = 0.4
[SRC_RADIUS] = 4
[SRC_AMPLITUDE] = 200
[NUM_STRUCTURES] = 5
[STRUCTURE] = 0
[P1] = 0.4 0.2
[P2] = 0.4 0.37
[P3] = 0.6 0.37
[P4] = 0.6 0.2
[STRUCTURE] = 1
[P1] = 0.4 0.43
[P2] = 0.4 0.6
[P3] = 0.6 0.6
[P4] = 0.6 0.43
[STRUCTURE] = 2
[P1] = 0.1 0.2
[P2] = 0.1 0.22
[P3] = 0.25 0.22
[P4] = 0.25 0.2
[STRUCTURE] = 3
```

```
[P1] = 0.1 0.58
[P2] = 0.1 0.6
[P3] = 0.25 0.6
[P4] = 0.25 0.58
[STRUCTURE] = 4
[P1] = 0.1 0.25
[P2] = 0.1 0.55
[P3] = 0.12 0.55
[P4] = 0.12 0.25
```

```
[SCENARIO] = 1
[OMP_THREADS] = 2
[SIZEX] = 300
[SIZEY] = 300
[H] = 0.25
[MAX_TIME] = 30
[TIME_STEP] = 0.05
[SAVE_TIME] = 10
[SRC_X] = 0.3
[SRC_Y] = 0.4
[SRC_RADIUS] = 4
[SRC_AMPLITUDE] = 200
[NUM_STRUCTURES] = 5
...
(structuri identice cu scenariul anterior)
...
```

```
[SCENARIO] = 2
[OMP_THREADS] = 2
[SIZEX] = 400
[SIZEY] = 400
[H] = 0.25
[MAX_TIME] = 30
[TIME_STEP] = 0.05
[SAVE_TIME] = 10
[SRC_X] = 0.3
[SRC_Y] = 0.4
[SRC_RADIUS] = 4
[SRC_AMPLITUDE] = 200
[NUM_STRUCTURES] = 5
...
(structuri identice cu scenariul anterior)
...
```

```
[SCENARIO] = 3
[OMP_THREADS] = 2
[SIZEX] = 500
[SIZEY] = 500
[H] = 0.25
[MAX_TIME] = 30
[TIME_STEP] = 0.05
```

```
[SAVE_TIME] = 10
[Src_X] = 0.3
[Src_Y] = 0.4
[Src_RADIUS] = 4
[Src_AMPLITUDE] = 200
[NUM_STRUCTURES] = 5
...
(structuri identice cu scenariul anterior)
...
```

```
[SCENARIO] = 4
[OMP_THREADS] = 2
[SIZEX] = 600
[SIZY] = 600
[H] = 0.25
[MAX_TIME] = 30
[TIME_STEP] = 0.05
[SAVE_TIME] = 10
[Src_X] = 0.3
[Src_Y] = 0.4
[Src_RADIUS] = 4
[Src_AMPLITUDE] = 200
[NUM_STRUCTURES] = 5
...
(structuri identice cu scenariul anterior)
...
```

```
[SCENARIO] = 5
[OMP_THREADS] = 2
[SIZEX] = 700
[SIZY] = 700
[H] = 0.25
[MAX_TIME] = 30
[TIME_STEP] = 0.05
[SAVE_TIME] = 10
[Src_X] = 0.3
[Src_Y] = 0.4
[Src_RADIUS] = 4
[Src_AMPLITUDE] = 200
[NUM_STRUCTURES] = 5
...
(structuri identice cu scenariul anterior)
...
```

```
[SCENARIO] = 6
[OMP_THREADS] = 2
[SIZEX] = 800
[SIZY] = 800
[H] = 0.25
[MAX_TIME] = 30
[TIME_STEP] = 0.05
```

```
[SAVE_TIME] = 10
[Src_X] = 0.3
[Src_Y] = 0.4
[Src_RADIUS] = 4
[Src_AMPLITUDE] = 200
[NUM_STRUCTURES] = 5
...
(structuri identice cu scenariul anterior)
...
```

```
[SCENARIO] = 7
[OMP_THREADS] = 2
[SIZEX] = 900
[SIZEY] = 900
[H] = 0.25
[MAX_TIME] = 30
[TIME_STEP] = 0.05
[SAVE_TIME] = 10
[Src_X] = 0.3
[Src_Y] = 0.4
[Src_RADIUS] = 4
[Src_AMPLITUDE] = 200
[NUM_STRUCTURES] = 5
...
(structuri identice cu scenariul anterior)
...
```

```
[SCENARIO] = 8
[OMP_THREADS] = 2
[SIZEX] = 1000
[SIZEY] = 1000
[H] = 0.25
[MAX_TIME] = 30
[TIME_STEP] = 0.05
[SAVE_TIME] = 10
[Src_X] = 0.3
[Src_Y] = 0.4
[Src_RADIUS] = 4
[Src_AMPLITUDE] = 200
[NUM_STRUCTURES] = 5
...
(structuri identice cu scenariul anterior)
...
```

În urma rulărilor s-au obținut următoarele rezultate, prezentate în figura (1) sub forma unui graf care descrie pe orizontală dimensiunea matricei de discretizare în număr de celule și pe orizontală timpul de rulare pentru fiecare scenariu în parte:

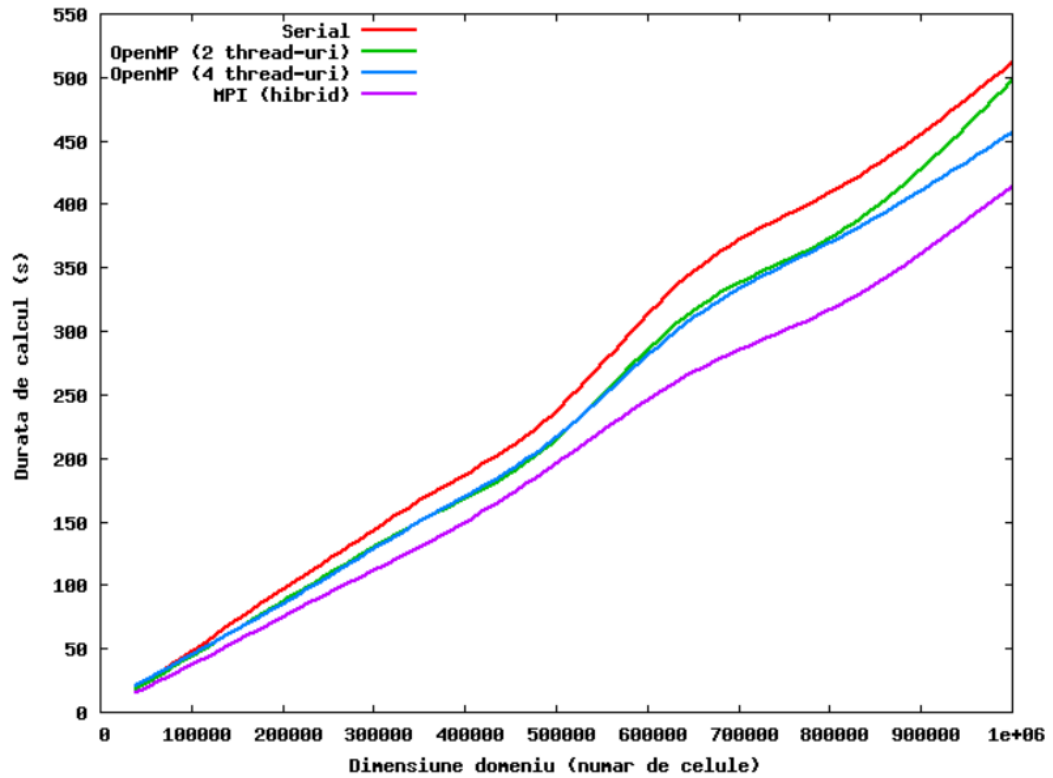


Figura 5.7: Analiza performanțelor

Astfel, pentru cazul de rulare de mai sus speed-up-ul obținut este descris în graficul de mai jos:

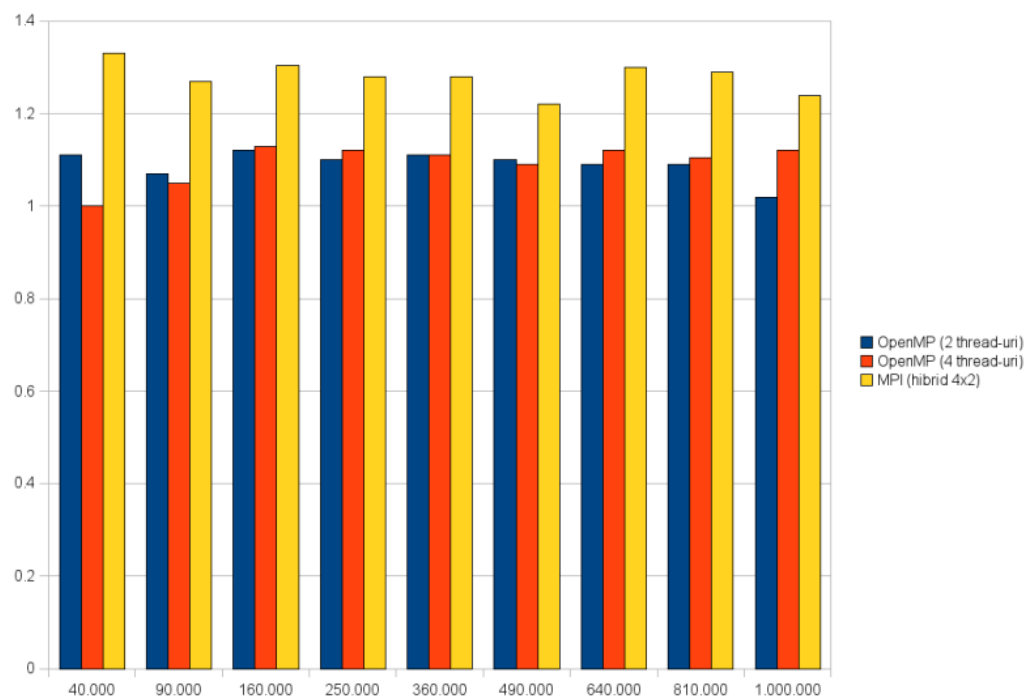


Figura 5.8: Speed-Up

Capitolul 6

Concluzie

Lucrarea de față încearcă în implementarea ei o abordare cât mai optimizată pentru simulările acustice folosind metoda numerică cu diferențe finite centrate calculate cu schema explicită centrată. Analiza codului serial, și apoi a celui paralelizat, a dus la efectuarea multor micro-optimizări ale codului care au redus, rând pe rând, durata totală de execuție. Compilarea eficientă aduc și ele un aport la creșterea performanțelor.

În plus față de optimizările aduse algoritmului de bază, paralelizarea codului folosind OpenMP a îmbunătățit timpii de calcul, speed-up-ul obținut fiind unul destul de bun, din testare și analiza performanțelor reieșind că se obțin timpi destul de buni chiar și pentru sisteme cu procesoare dual-core (cele care domină piața la momentul actual).

Implementarea hibridă MPI-OpenMP a sporit și mai mult performanțele în special pentru domenii mari și foarte mari de calcul. Utilizarea mai multor noduri de calcul în mod distribuit (cu calculul pe fiecare nod executat în paralel) a dus la posibilitatea folosirii unor domenii de calcul mult mai mari fără a fi afectate performanțele de problemele celorlalte 2 tipuri de implementări, cum ar fi depășirea spațiului maxim din cache-ul procesorului.

Aplicația este de asemenea pusă în valoare și de modalitatea foarte permisivă și complexă de a furniza datele de intrare precum și formatul flexibil folosit pentru rezultate, format de ieșire deseori folosit în aplicațiile de fizică computațională.

Ca direcție viitoare de dezvoltare, aplicația ar putea fi extinsă la metoda elementelor finite cuplată cu un program de discretizare a spațiului fizic (care constituie o problemă complexă și mare consumatoare de resurse la fel ca și aplicația în sine). În plus, discretizarea ar putea transforma din domeniu fizic structuri mult mai complexe din punct de vedere geometric. De asemenea, alte fenomene acustice ar putea fi puse în evidență folosind MEF, cum ar fi, spre exemplu, absorbția sau efectul Doppler.

Cuprins

1	Introducere	2
2	Aspecte teoretice	4
2.1	Noțiuni generale fizice	4
2.1.1	Ecuția undei	8
2.1.2	Unda plană	9
2.1.3	Unda sferică	14
2.1.4	Reflexia	17
2.1.5	Acustica spațiilor închise. Unde acustice neamortizate	18
2.2	Metode numerice de rezolvare	23
3	Implementare	27
3.1	Inițializarea parametrilor de rulare	27
3.2	Algoritmul serial	29
3.3	Paralelizare cu OpenMP	31
3.4	Paralelizare și calcul distribuit cu MPI	32
3.5	Salvarea datelor în fișier	36
4	Testare	37
4.1	Arhitectura sistemului de testare	37
4.2	Rezultate obținute	39
5	Performanțe	44
5.1	Profiling	44

5.2	Timpi de rulare și speed-up	49
6	Concluzie	55

Bibliografie

- [1] Ioan Dumitrescu – *Simularea Câmpurilor potențiale*, 1983, Editura Academiei RS Romania București
- [2] Thomas D. Rossing and Neville H. Fletcher, *Principles of vibration and sound*, 1995, Springer New York
- [3] Nicolae Enescu, Ioan Magheț, Mircea Alexandru Sârbu *Acustica tehnică*, 1998, Editura ICPE
- [4] A. Quarteroni and A. Valli, *Numerical Approximation of Partial Differential Equations*, 1998, Springer-Verlag
- [5] Quinn, M. J. , *Parallel Programming in C with MPI and OpenMP*, 2003 , McGraw Hill Higher Education
- [6] Gropp, W., Lusk, E. Skjellum, A., *Using MPI, Portable Parallel Programming with the Message-Passing Interface*, 1998, Cambridge, MA MIT PRESS, 2nd edition
- [7] Smith, L. and Bull, M., *Development of mixed mode MPI / OpenMP applications*, 2001, Sci. Program. 9, 2,3 (Aug. 2001), 83-98
- [8] Alexandru I. Schiop, *Metode numerice pentru rezolvarea ecuațiilor diferențiale*, 1975, Editura Academiei R.S.R
- [9] <http://www.open-mpi.org/>
- [10] <http://www.paraview.org>

[11] <http://gridengine.sunsource.net/howto/howto.html>

[12] http://developers.sun.com/solaris/articles/analyzer_qs.html