KLAIPĖDOS UNIVERSITETAS
GAMTOS IR MATEMATIKOS MOKSLŲ FAKULTETAS
INFORMATIKOS KATEDRA

LINA BLIŪDŽIUTĖ

GMIN03 grupės studentė

LYGIAGRETAUS SKAIČIAVIMO TECHNOLOGIJŲ NAUDOJIMAS KURŠIŲ MARIŲ EKOLOGINIAME MODELYJE

Baigiamasis magistro darbas

Darbo vadovas doc. dr. Artūras Razinkovas

Darbo konsultantas doc. dr. Petras Zemlys

KLAIPĖDA, 2005

ANOTACIJA

Šiuolaikiniai kompiuteriai yra pajėgūs per pakankamai trumpą laiką išspręsti daugelį uždavinių, tačiau yra sričių, kur skaičiavimai gali užrukti mėnesius ar net metus. Lygiagretieji algoritmai yra vienas iš būdų pagreitinti ilgai trunkančius sudėtingus skaičiavimus.

Darbe nagrinėjamos lygiagrečių skaičiavimų bendros atminties (OpenMP) ir paskirstytos atminties (MPI) technologijos, jų architektūros, privalumai ir trūkumai. Kuršių marių modelio programiniame kode nustatomos potencialios lygiagretinimo vietos, kuriose pritaikomos lygiagrečių skaičiavimų technologijos: OpenMP ir MPI. Atliekama modelio greitaeigiškumo analizė.

PAGRINDINIAI ŽODŽIAI: lygiagretūs skaičiavimai, OpenMP, MPI, SHYFEM, greitinimas, Amdahl'o dėsnis.

SUMMARY

Modern computers are capable of completing most of the tasks in fairly short time, however there are areas in what calculations can last for months and even years. Parallel algorithms are one of the ways to accelerate long-lasting calculations.

In the thesis we analyze parallel computing technologies OpenMP (shared memory) and MPI (distributed memory), cons and pros of their architecture. We identify potential parts of program code of Curonian lagoon model for parallelizing, in which chosen parallel computing technologies OpenMP and MPI is applied. Also we make the runtime speedup analysis.

KEY WORDS: parallel computing, OpenMP, MPI, SHYFEM, speedup, Amdahl's low.

TERMINŲ IR SANTRUMPŲ ŽODYNĖLIS

CPU (central processing unit) – procesorius.

I/O (input/output) – įvestis/išvestis.

Klasteris – tai kompiuterių rinkinys, skirtas lygiagretiems uždaviniams atlikti. [1]

MPI (Message Passing Interface) – tai komunikacinių paprogramių bibliotekos (MPL) standartas. [2]

MPP (massively parallel processing) – atskirų CPUs lygiagretus procesorinis apdorojimas.

OP (operating system) – operacinė sistema.

OpenMP – standartas, aprašantis kompiliatoriaus direktyvas, bibliotekos paprogrames ir aplinkos kintamuosius, kurie gali būti naudojami bendros atminties lygiagretinimui Fortrano ir C/C++ programose. [5]

PC (personal computer) – asmeninis kompiuteris.

PVM (Parallel Virtual Machine) – tai programinis paketas, leidžiantis per tinklą sujungti įvairius kompiuterius ir naudoti juos kaip vieną didelį lygiagretųjį kompiuterį. [2]

SHYFEM (shallow water hydrodynamic finite element model) – seklių telkinių hidrodinaminis baigtinių elementų modelis.

SMP (Symmetric Multiprocessing) – simetrinis daugiaprocesorinis apdorojimas.

TURINYS

ĮVADAS

Šiuolaikiniai kompiuteriai yra pajėgūs per pakankamai trumpą laiką išspręsti daugelį uždavinių, tačiau yra sričių, kur skaičiavimai gali užrukti mėnesius ar net metus. Sprendžiant tokius uždavinius labai svarbų vaidmenį atlieka lygiagretūs skaičiavimai.

Klaipėdos universiteto Baltijos pajūrio aplinkos planavimo ir tyrimo institute yra vykdomi Kuršių marių ekologinio modelio kūrimo darbai (modelis SHYFEM – angl. Finite Element Model for Coastal Seas). Modelis yra erdvinis, jame sprendžiamos diferencialinės lygtys dalinėmis išvestinėmis, todėl greitaeigiškumo klausimas yra labai aktualus.

Šiame darbe nagrinėjamos lygiagrečių skaičiavimų technologijos, jų architektūros, privalumai ir trūkumai, nustatomos potencialios Kuršių marių modelio programinio kodo lygiagretinimo vietos, kuriose pritaikoma pasirinktos lygiagrečių skaičiavimų technologijos, bei atliekama modelio greitaeigiškumo analizė.

Darbo tikslas: Modelio SHYFEM kompiuterinės programos greitaeigiškumo optimizavimas, taikant lygiagrečių skaičiavimų metodus.

Uždaviniai:

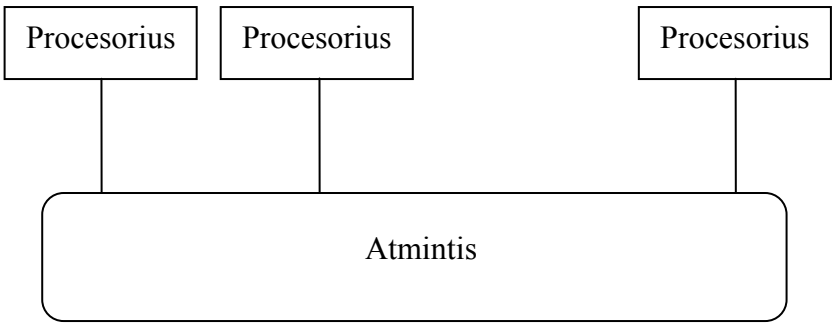
1. Išnagrinėti lygiagrečių skaičiavimų realizavimo būdus;
2. Išanalizuoti SHYFEM programinį kodą ir nustatyti potencialias lygiagretinimo vietas;
3. Pasirinkti ir pritaikyti lygiagrečių skaičiavimų technologijas;
4. Įvertinti programos darbo pagreitėjimą.

1. LYGIAGREČIŲ SKAIČIAVIMŲ TECHNOLOGIJOS

Lygiagretieji algoritmai yra vienas iš būdų pagreitinti ilgai trunkančius sudėtingus skaičiavimus. Lygiagretinant programos kodą pirmiausia kyla klausimai, kaip išskaidyti užduotį į nepriklausomas dalis, kiek procesų reikės panaudoti, kaip paskirstyti užduotis procesams, kokią tinkamiausią technologiją pasirinkti lygiagretaus skaičiavimo realizavimui. Šiame skyriuje apžvelgsime lygiagrečių skaičiavimų technologijas, jų architektūras, privalumus ir trūkumus.

1.1. Simetrinės technologijos (bendros atminties paradigma)

Simetrinės technologijos naudojamos programų paleidimui viename kompiuteryje: algoritmas išskaidomas į lygiagrečiai vykdomas dalis (angl. threads), kurių skaičius (pagal nutylėjimą) atitinka CPU skaičių. Procesai naudoja bendrą atmintį (*1 pav.*).



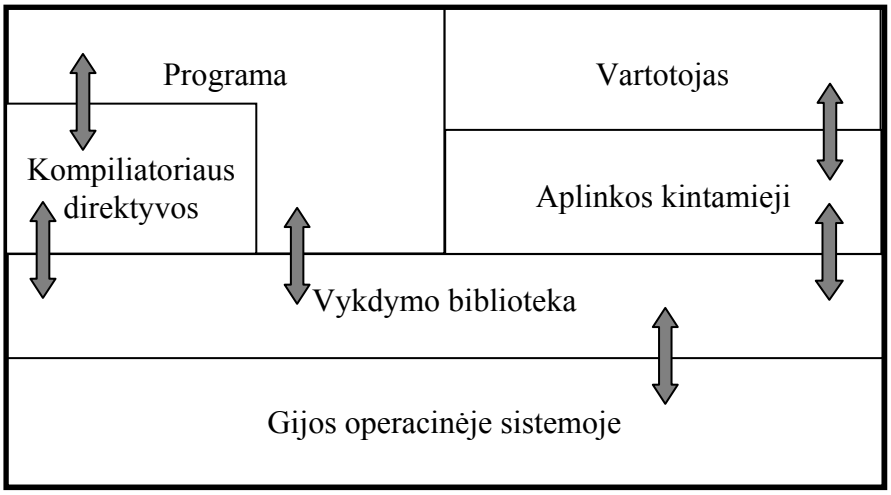
1 pav. Bendros atminties paradigma lygiagretiems skaičiavimams

Vienas iš populiariausių ir standartizuotų bendros atminties paradigmos metodų yra OpenMP.

1.1.1. OpenMP standartas

OpenMP yra realizuojamas kompiliatoriaus lygmenyje specialių direktyvų pagalba, kurios yra įterpiamos į Fortrano arba C kodą kaip komentarai. *2 pav.* pateikta OpenMP architektūra. Pagrindiniai OpenMP elementai [11]:

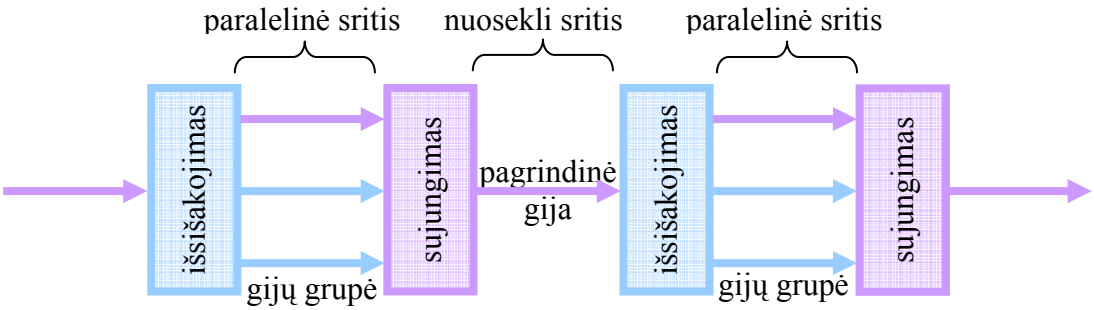
- Direktyvos: paralelinės srities, darbo pasidalinimo, sinchronizacijos;
- Bibliotekos paprogramės: omp_set_num_thread, omp_get_num_thread, omp_get_max_threads, omp_get_thread_num, omp_get_num_procs, omp_in_parallel, omp_set_dynamic, omp_get_dynamic, omp_set_nested, omp_get_nested, omp_init_lock, omp_destroy_lock, omp_set_lock, omp_unset_lock, omp_test_lock
- Aplinkos kintamieji: omp_shedule, omp_num_threads, omp_dynamic, omp_nested.



2 pav. OpenMP architektūra [7]

1.1.1.1. OpenMP direktyvos

Nuoseklioje srityje vykdomas algoritmas paralelinėje srityje išskaidomas į lygiagrečiai vykdomas dalis. Paralelinės ir nuoseklios srities sąvokas galima pailiustruoti išsišakojimo-sujungimo (angl. fork-join) schema (*3 pav.*).



3 pav. Išsišakojimo-sujungimo modelis [7]

Visos OpenMP programos prasideda kaip vienas procesas: pagrindinė gija (angl. master thread) vykdo kodą, kol sutinkamas paralelinės srities (angl. parallel region) direktyva. Pagrindinė gija sukuria gijų grupę paralelinės srities paleidimo metu (išsišakojimas). Programos sakiniai, kurie priklauso paralelinės srities direktyvai, yra vykdomi gijų grupėje lygiagrečiai. Kai gijų grupė užbaigia vykdyti sakinius paralelinės srities direktyvoje, gijos yra sinchronizuojamos ir nutraukiamos, paliekant tik pagrindinę giją (sujungimas). [9] Paralelinės srities direktyvos panaudojimo pavyzdys Fortrano kalboje [7]:

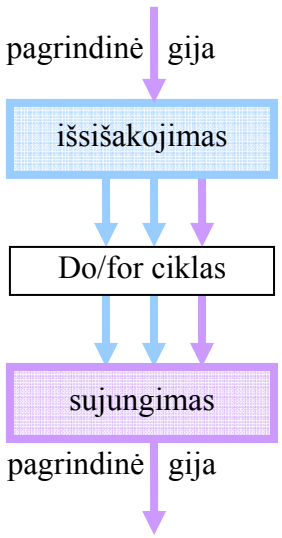
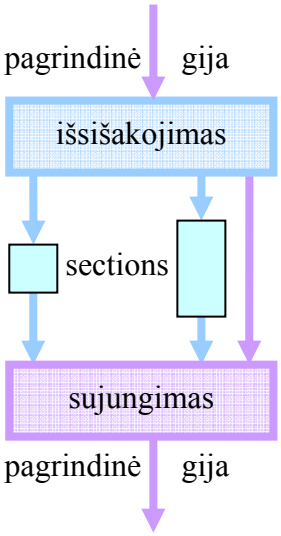
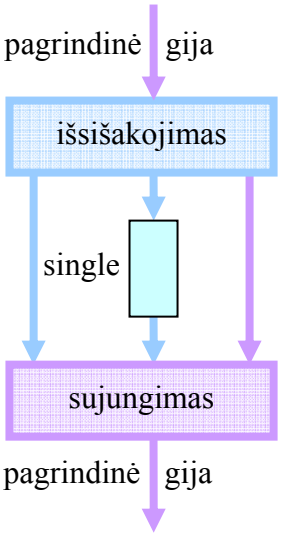
```
!$OMP PARALLEL
!$OMP DO
!$OMP&PRIVATE(I)
```

```
      DO I=1,n
        A(I)= A(I) * A(I) - 3
      END DO
!$OMP END DO NOWAIT
!$OMP END PARALLEL
```

Kintamieji gali būti privatūs ir bendri. Direktyvos parametras <private(kintamųjų sąrašas)> skelbia, kad besikeičiančio kintamojo kopija sukuriama kiekvienoje gijoje. Parametras <shared(kintamųjų sąrašas)> skelbia, kad visos gijos prieis prie kintamojo, sukurto pagrindinėje gijoje. [7] Jeigu kintamieji neapibrėžti, pagal nutylėjimą jie yra bendro naudojimo, išskyrus lokalius kintamuosius paralelinėje srityje bei kintamuosius DO cikle. [6]

Kita direktyvų grupė – darbo pasidalinimo direktyvos (*1 lentelė.*), kurios paskirsto nurodyto kodo srities vykdymą tarp gijų.

1 lentelė. OpenMP darbo pasidalinimo direktyvos [9]

Pavadinimas	DO / for	SECTIONS	SINGLE
Aprašymas	dalina ciklo iteracijas tarp gijų	suskirsto darbą į atskiras, diskrečias dalis, kiekvieną dalį vykdo atskiros gijos	priskiria vykdymo bloką vienai gijai
Schema			

Galimi du kombinuoti lygiagretinimo būdai:

- naudojant paralelinės srities direktyvą, sekcijų direktyvą ir DO ciklą,

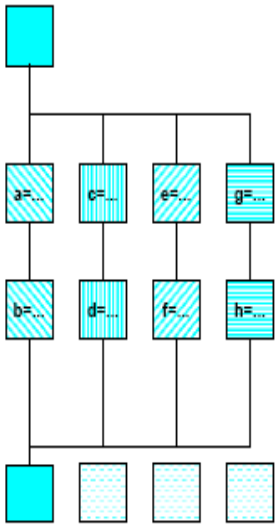
pavyzdys:

```
!$omp parallel shared(A,B)private(i)
        !$omp sections
                !$omp section
                    !$omp do
                        do i=1,n
                            A(i) = A(i) * A(i) - 3.
                        end do
                    !$omp end do
                !$omp section
                    !$omp do
                        do i=1,n
                            B(i) = B(i) * B(i) + 5.
                        end do
                    !$omp end do nowait
        !$omp end sections
!$omp end parallel
```

- naudojant paralelinės srities direktyvą ir sekcijų (angl. *sections*) direktyvoje esančias sekcijas (*4* ***pav.***),

pavyzdys:

```
!$OMP PARALLEL
        !$OMP SECTIONS
          { a=...;
          b=...; }
          !$OMP SECTION
                    { c=...;
                    d=...; }
          !$OMP SECTIONS
                    { e=...;
                    f=...; }
          !$OMP SECTIONS
                    { g=...;
                    h=...; }
        !$OMP END SECTIONS
!$OMP END PARALLEL
```



4 pav. Sekcijų direktyvos pavyzdys

2 lentelė. pateikta trečioji direktyvų grupė – sinchronizacijos direktyvos.

2 lentelė. Sinchronizacijos direktyvos [9]

Direktyvos	MASTER (Pagrindinė)	CRITICAL (Kritinė)	BARRIER (Barjerinė)	ATOMIC (Atominė)	FLUSH (Įrašymo)	ORDERED (Sutvarkyta)
Tikslai	Apibrėžia kodo sritį, kurią vykdo tik pagrindinė gija. Visos kitos gijos šią kodo dalį praleidžia.	Apibrėžia kodo sritį, kurią vykdo tik viena gija bet kuriuo laiku.	Sinchronizuoja visas gijas grupėje; kai BARRIER direktyva yra pasiekiama, gija lauks kol kitos gijos pasieks barjerą, tada visos gijos tęs kodo vykdymą lygiagrečiai.	Nurodo, kad konkreti atminties vieta turi būti keičiama atomiškai (tik viena gyja vienu metu).	Identifikuoja sinchronizacijos tašką, kuriame programa turi suformuoti pilną atminties vaizdą. Gijoms matomi kintamieji šiame taške įrašomi į atmintį.	Apibrėžia kodo sritį, kurioje esančio ciklo iteracijos bus vykdomos ta pačia tvarka kaip ir tuo atveju, jei jos būtų vykdomos nuosekliu režimu.
Formatas (Fortanas)	!$OMP MASTER *blokas* !$OMP END MASTER	!$OMP CRITICAL [vardas] *blokas* !$OMP END CRITICAL	!$OMP BARRIER	!$OMP ATOMIC *sakinio išraiška*	!$OMP FLUSH [sąrašas]	!$OMP ORDERED *blokas* !$OMP END ORDERED
Pastaba		Jeigu gija vykdo kodą kritinės srities viduje, o kita gija siekia prisijungti prie šios srities ir stengiasi įvykdyti kodą, pastarosios gijos veiksmas bus blokuojamas, kol pirmoji gija išeis iš kritinės srities. Laisvai pasirenkami vardai leidžia egzistuoti skirtingoms kritinėms sritims. Vardai veikia kaip globaliniai identifikatoriai. Skirtingos kritinės			Sąrašas kintamųjų, kurie bus įrašyti, sudaromas tam, kad išvengti visų kintamųjų įrašymo.	Gali pasirodyti tik dinaminėse DO ar parallel DO realizacijose. Tik viena gija yra leidžiama sutvarkytoje dalyje vienu metu. Bet kokia ciklo iteracija negali vykdyti sutvarkytos dalies daugiau nei vieną kartą. Ciklas, kuris turi sutvarkytą direktyvą, turi būti ciklas su ORDERED parametru.

Direktyvos	MASTER (Pagrindinė)	CRITICAL (Kritinė)	BARRIER (Barjerinė)	ATOMIC (Atominė)	FLUSH (Įrašymo)	ORDERED (Sutvarkyta)
		sritys su tais pačiais vardais laikomos kaip viena ir ta pati sritis. Visos kritinės sritys, kurios yra nepavadintos, yra laikomos kaip viena sritis.				
Apribojimai	Vykdymo metu draudžiama įeiti į MASTER bloką ar išeiti iš jo.	Vykdymo metu draudžiama įeiti į CRITICAL bloką ar išeiti iš jo.	Privalo būti pasiektas visų gijų grupėje arba nei vienos gijos.	Taikoma tik vienam sekančiam teiginiui.		

1.1.1.2. OpenMP bibliotekos paprogramės

Bibliotekos paprogramės yra skirtos gijų valdymui. Jos gali būti skirstomos į tris kategorijas [7]:

- užklausti ir nustatyti daugiasrautinį režimą:
  - gauti/nustatyti gijų ar procesorių skaičių (omp_set_num_threads, omp_get_num_threads, omp_in_parallel);
  - gauti gijos identifikacinį numerį (omp_get_thread_num). Identifikacinis numeris gali būti nuo nulio iki omp_get_num_threads()-1 imtinai. Pagrindinės gijos identifikacinis numeris grupėje yra nulis.
- nustatyti ir gauti vykdymo aplinką:
  - užklausti/nustatyti vidinį (angl. nested) lygiagretinimą (omp_get_nested, omp_set_nested);
  - užklausti/nustatyti dinaminį gijų skaičių skirtingose paralelinėse srityse (omp_get_dynamic, omp_set_dynamic);
- manipuliuoti blokavimus (angl. lock):
  - blokavimo kintamasis atlieka gijos sinchronizavimą (omp_init_lock, omp_destroy_lock);
  - blokavimo paprogramės (omp_init_lock, omp_set_lock, omp_unset_lock).

1.1.1.2. OpenMP aplinkos kintamieji

Lygiagrečios programos elgsenai kontroliuoti vykdymo metu naudojami *aplinkos kintamieji* (*3 lentelė.*), kuriuos programuotojas bet kada gali perrašyti.

3 lentelė. OpenMP Aplinkos kintamieji [7]

Pavadinimas	Aprašymas	Reikšmė
OMP_NUM_THREADS	Nusako gijų, kurios bus naudojamas programos vykdymo metu, skaičių. Kai neapibrėžtas gijų skaičius, aplinkos kintamojo reikšmė yra maksimalus jų skaičius.	teigiamas skaičius
OMP_DYNAMIC	Leidžia ar uždraudžia dinamiškai keisti gijų skaičių, prieinamų paralelinių sričių vykdymui.	TRUE ar FALSE
OMP_NESTED	Leidžia ar uždraudžia vidinį lygiagretinimą.	TRUE ar FALSE
OMP_SCHEDULE	Prieinamas tik DO ir PARALLEL DO direktyvoms su paskirstymo (shedule) parametru nustatytu į RUNTIME. Nustato kaip ciklo iteracijos paskirstomos tarp procesorių.	“static, 2“, "guided, 4", "dynamic"

Taigi OpenMP gali būti pasirinktas dėl kelių priežasčių:

- direktyvų įterpimas reikalauja minimalių pastangų perprogramuojant kodą;

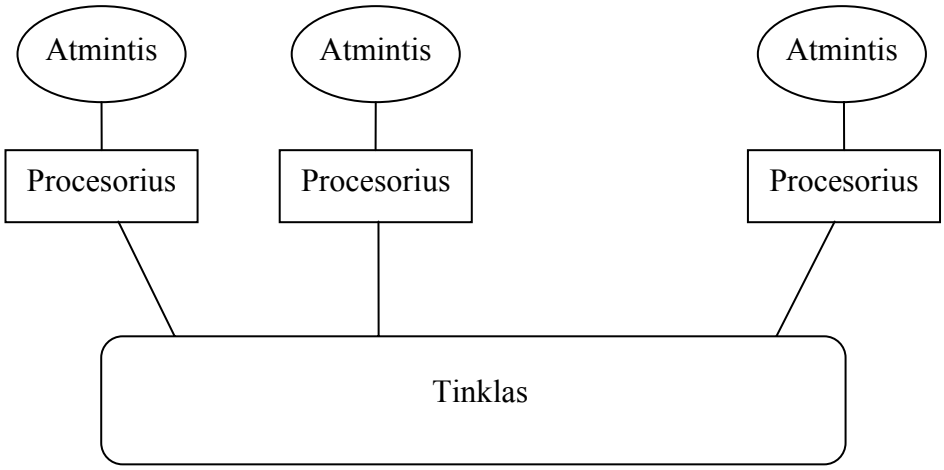
- atjungus kompiliatoriaus -openmp opciją, programa vykdoma nuosekliai;
- dėl bendros atminties naudojimo nereikalingas duomenų apsikeitimas.

OpenMP trūkumai:

- OpenMP programos vykdomos tik atskiruose kompiuteriuose, todėl kartais gali nepakakti procesorių skaičiaus;
- Daugiaprocesoriniai superkompiuteriai yra labai brangūs.

1.2. Asimetrinės technologijos (paskirstytos atminties paradigma)

Asimetrinės technologijos lygiagretiems skaičiavimams naudoja sujungtų į tinklą kompiuterių (klasterių), procesorius ir atmintį (*5 pav.*), todėl būtinas duomenų apsikeitimas tarp procesų.



5 pav. Paskirstytos atminties paradigma lygiagretiems skaičiavimams

Šiuo metu paskirstytos atminties (pranešimų siuntimo) paradigma tampa vis daugiau ir daugiau populiari. Viena iš priežasčių yra platus platformų, kurios palaiko pranešimų siuntimo modelį, paplitimas. Programas parašytas šiuo metodu galima paleisti naudojant kelių procesorių: paskirstytos atminties daugiaprocesorines arba kelias vienprocesorines (sujungtas į tinklą) sistemas. Pranešimų siuntimo paradigmoje procesai komunikuoja siųsdami vienas kitam pranešimus. [8]

Prie paskirstytos atminties technologijų priskiriamos MPI, PVM.

1.2.1. Pranešimų perdavimo interfeisas (MPI)

MPI – tai komunikacinių paprogramių bibliotekos (MPL) specifikacija, kuria yra naudojamasi programuojant C ir Fortran kalbomis. Biblioteką sudaro funkcijų ir paprogramių rinkinys, užtikrinantis komunikaciją tarp procesų. [2]

MPI standartas apibrėžia tiesiogines (angl. point to point) komunikacijas, kai komunikacija vyksta tik tarp dviejų procesų, ir kolektyvines (angl. collective) komunikacijas, kai komunikacijos ar sinchronizacijos operacijos įtraukia procesų grupes. [10]

MPI gali būti įvairiai realizuotas, pvz. MPICH, LAM-MPI ir kt. bibliotekomis, atitinkančiomis šį standatrtą. Kompiliatoriai, naudojantys tiek vieną, tiek kitą biblioteką, Fortrano 77 programose iškviečiami ta pačia komanda *mpif77*.

Visi MPI paprogramių ir konstantų vardai prasideda su prefiksu MPI_, kad išvengti pavadinimų kolizijų. [8] Pagrindinės MPI funkcijos: *MPI_Init* (inicijuoja lygiagrečiojo algoritmo darbo pradžią), *MPI_Finalize* (paskelbia lygiagrečiojo algoritmo pabaigą), *MPI_Send* (naudojama duomenų siuntimui) ir *MPI_Recv* (naudojama duomenų gavimui). [4] Pastarosios dvi funkcijos įvykdomos tada, kai vienas procesas siunčia duomenis (*MPI_Send*), o kitas pasiruošęs juos priimti (*MPI_Recv*), t. y. tuo metu nevykdo jokių kitų veiksmų. [8]

Pirmoji MPI procedūra iškviečiama bet kokioje MPI programoje turi būti inicializavimo procedūra MPI_INIT. Kiekviena MPI programa turi išsikviesti šią procedūrą vieną kartą, prieš bet kokias kitas MPI procedūras. MPI_FINALIZE procedūra iškviečiama programos pabaigoje. Ši procedūra išvalo visas MPI duomenų struktūras ir po jos iškvietimo jokia kita procedūra negali būti iškviesta. Visos MPI programos turi įtraukti standartinį parametrų failą *mpif.h* (Fortran), kuris reikalingas dažnai naudojamų kintamųjų paskelbimui. Programos pavyzdys Fortrano kalboje [8]:

```
include 'mpif.h'
integer errcode
c MPI inicializavimas
call MPI_INIT (errcode)
c pagrindinė programos dalis
 ....
c MPI užbaigimas
call MPI_FINALIZE (errcode)
end
```

Svarbų vaidmenį MPI komunikacijose atlieka komunikatorius – procesų grupė. MPI_COMM-WORLD komunikatorius yra apibrėžtas iš anksto. Jį sudaro visi procesai. Kitus

komunikatorius gali sukurti pats vartotojas iš jau egzistuojančių komunikatorių, juos sujungiant ar išskaidant. Kiekvienas komunikatorius turi procesus, kurie komunikatoriaus viduje numeruojami nuo 0 iki N-1, kur N – procesų skaičius komunikatoriuje. Kiekvieno proceso numeris suprantamas kaip jo rangas (angl. rank). Rangas identifikuoja kiekvieną procesą komunikatoriaus viduje. Pavyzdžiui, rangas gali specifikuoti pranešimo šaltinį ar pristatymo vietą. Vartotojas bet kada gali sužinoti procesorių skaičių komunikatoriuje naudodamas *MPI_COMM_SIZE (comm, size)* paprogramę ir proceso rangą komunikatoriuje – *MPI_COMM_RANK (comm, rank)* paprogramę, kur *comm* yra komunikatoriaus pavadinimas. [8]

Pranešimo pagalba perkeliamos kintamųjų reikšmės. Visi MPI pranešimai yra nustatyto tipo (*4 lentelė.*), todėl turinio tipas turi būti vienodas išsiuntimo ir gavimo metu. Kai pranešimas išsiųstas, gavimo procesas turi tikėtis gauti tą patį duomenų tipą. Pavyzdžiui, jei procesas siunčia pranešimą su MPI_INTEGER duomenų tipu, gavimo procesas turi gauti duomenų tipą MPI_INTEGER, priešingu atveju komunikacija bus ne tiksli ir funkcionalumas neapibrėžtas. Išimtis yra tik su MPI_PACKED, kuris gali suderinti bet kokius tipus. Panašiai yra su Fortrano ar C/C++ kintamųjų tipais pranešime. Kintamieji turi atitikti MPI duomenų tipą, t.y. jei procesas siunčia pranešimą su duomenų tipu MPI_INTEGER, tai procesas kintamuosius turi apibrėžti INTEGER tipo, priešingu atveju funkcionalumas yra neapibrėžtas. Išimtis yra su MPI_BYTE ir MPI_PACKED

4 lentelė. MPI duomenų tipai

MPI duomenų tipas	FORTRAN duomenų tipas
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
MPI_BYTE	
MPI_PACKED	

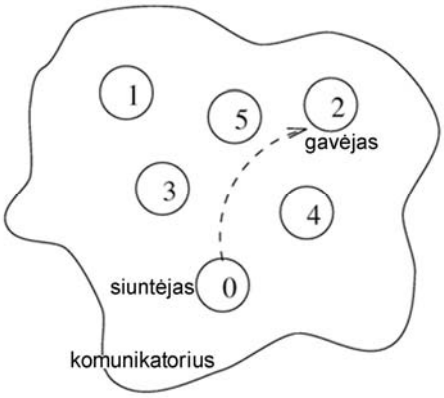
Pagrindiniai MPI duomenų tipai atitinka pagrindinius Fortrano duomenų tipus, išskyrus MPI_BYTE ir MPI_PACKED. Dauguma sudėtinių duomenų tipų gali būti sudaryti vykdymo metu. Šie duomenų tipai vadinami išvestiniais ir yra sudaryti iš pagrindinių duomenų tipų. Išvestų duomenų tipo aprašymas atliekamas per dvi stadijas [8]:

- duomenų tipo sudarymas iš egzistuojančių duomenų tipų naudojant kvietimą ar rekursinius kvietimus procedūroms: MPI_TYPE_CONTIGUOUS, MPI_TYPE_VECTOR, MPI_TYPE_HVECTOR, MPI_TYPE_INDEXED MPI_TYPE_HINDEXED, MPI_TYPE_STRUCT.

- naujas duomenų tipas – fiksuotas (angl. committed) su kreipimuosi į MPI_TYPE_COMMIT.

1.1.2.1. MPI tiesioginė komunikacija

Tiesioginė komunikacija apima tik du procesus. Vienas procesas siunčia pranešimą kitam (*6 pav.*). Tuo tiesioginė komunikacija skiriasi nuo kolektyvinės komunikacijos, kuri apima visą procesų grupę vienu metu.



6 pav. Tiesioginė komunikacija

Kad pasiųsti pranešimą, procesas "siuntėjas" iškviečia MPI_SEND, kuris nustato proceso "gavėjo" rangą komunikatoriuje. Pastarasis procesas iškviečia MPI_Recv, kad gauti pranešimą.
MPI_Send(message, count, datatype, dest, tag, comm, ierror), kur *message* – siunčiamas kintamasis, *count* – siunčiamų kintamųjų skaičius, *datatype* – MPI duomenų tipas, *dest* – proceso "gavėjo" rangas, *tag* – markeris, naudojamas išskirti pranešimo tipus, *comm* – komunikatorius, *ierror* – gražinimo reikšmė.
MPI_Recv(message, count, datatype, source, tag, comm, status, ierror), kur *source* – proceso "siuntėjo" rangas, *status* – gražinimo informacija.

Tiesioginėje komunikacijoje yra keturi siuntimo režimai (*5 lentelė.*). Šie režimai naudojami tiek blokavimo formose, tiek formose be blokavimo. [8]

5 lentelė. Komunikacijos režimai

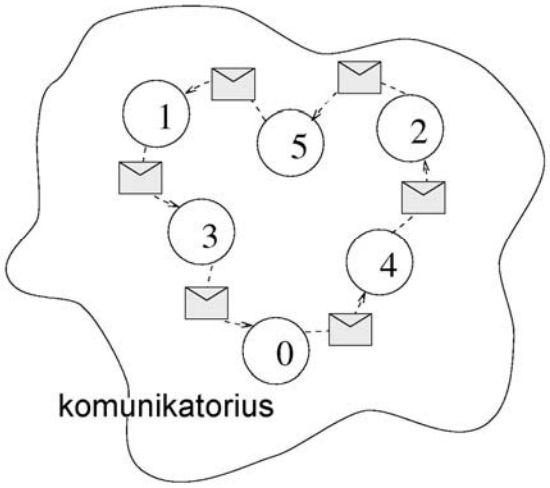
Režimai	Užbaigimo sąlygos	Procedūra
Sinchronizuotas (synchronous) siuntimas	baigiasi, kai gavimas yra užbaigtas	MPI_SSEND
Buferinis (buffered) siuntimas	visada baigiasi (jei neaptikta klaida) nepriklausomai ar buvo užbaigtas gaviams	MPI_BSEND
Standartinis (Standard) siuntimas	baigiasi, (jei neaptikta klaida) nepriklausomai ar gavimas yra užbaigtas	MPI_SEND
Pasiruošęs (ready) siuntimas	baigiasi, (jei neaptikta klaida) nepriklausomai ar gavimas yra užbaigtas	MPI_RSEND
Gavimo	baigiasi, kai pranešimas yra gautas	MPI_RECV

Siuntimo užbaigimas reiškia, kad siuntimo buferis gali būti saugiai panaudotas iš naujo. Standartinis, sinchronizuotas ir buferinis siuntimas skiriasi tik vienu atžvilgiu, kaip siuntimo užbaigimas priklauso nuo pranešimo gavimo.

MPI tiesioginės komunikacijos savybės [8]:

1. *Pranešimų eilės tvarkos išsaugojimas*. Pranešimai ne pralenkia vienas kito. Tarkim, procesas A siunčia du pranešimus procesui B su tuo pačiu komunikatoriumi. Procesas B siunčia du gavimo kvietimus, kurie suderina abu siuntimus. Tuomet du pranešimai bus gauti ta tvarka, kuria buvo išsiųsti (first in first out).
2. *Nėra galimybės siųsti ir gauti pora pranešimų iškarto*. Jei vienas MPI procesas siunčia pranešimą, o kitas procesas siunčia gavimo derinimą, tuomet bet kuris siuntimas ar gavimas ilgainiui užsitęs. Galimi du scenarijai: siuntimą gauna trečias procesas su suderintu gavimu, kurio atveju siuntimas užbaigiamas, bet antro proceso gavimas neužbaigiamas, arba trečias procesas pasiunčia pranešimą, kurį gauna antras procesas, kurio atveju gavimas užbaigiamas, bet pirmo proceso siuntimas neužbaigiamas.

Blokavimo komunikacija negrąžina pranešimo, kol komunikacija neužbaigta, todėl gali atsitikti siuntimo situacija, kai kiekvienas procesas siunčia pranešimą kitam procesui. Priklausomai nuo detalių įgyvendinimo siuntimas gali būti neužbaigtas, kol gavimas prasidėjo. Taigi kiekvienas procesas siunčia ir nei vienas negauna, galima aklavietė (*7 pav.*) ir nei viena iš komunikacijų bus nebaigta.



7 pav. Aklavietė (Deadlock)

Galimas sprendimas [8]: komunikacija pasirenka tarkim nelyginį procesą siuntimui ir lyginį procesą gavimui, bet be aklavietės problemos yra dar viena – komunikacijos lėtumas dėl komunikacijos tinklo ir proceso, esančio kitame komunikacijos gale. Komunikacijos blokavimo metu, procesas laukia, kol kiekviena komunikacija atsilaisvins. Ne blokavimo komunikacija yra

vienas iš metodu išspręsti šias problemas, nes blokavimo metu procesai iškviečia MPI procedūras įvykdyti komunikaciją (siuntimą ar gavimą), bet procedūra gražinama anksčiau nei komunikacija užbaigiama. Komunikacija užbaigiama nuošalyje, o procesas gali toliau vykdyti kitą darbą, grįždamas vėliau patikrinti, ar komunikacija užbaigta sėkmingai, todėl komunikacija yra suskirstyta į dvi operacijas: pradėjimo ir užbaigimo testavimo.

Neblokuojančios tiesioginės (angl. point-to-point) funkcijos yra dviejų dalių: skiria operaciją ir laukia rezultato, taip pat įtraukia apklausos/testo pasirinkimą, t.y. tikrina, ar operacija pabaigta. [10] Kai naudojama neblokavimo komunikacija, svarbu įsitikinti, kad komunikacija užbaigta anksčiau negu komunikacijos rezultatai bus naudojami ar komunikacijos buferis bus panaudotas iš naujo. Galimi du užbaigimo testavimo tipai: laukimo (angl. wait) ir testo (angl. test). Pirmuoju atveju procedūros blokuojamos iki komunikacijos užbaigimo. Tai yra naudinga, kai komunikacijos duomenys yra reikalingi skaičiavimams ar komunikacijos buferiui, kur ketinamas panaudoti iš naujo, todėl neblokavimo komunikacija šiuo atveju atitinka blokavimo komunikaciją. Antruoju atveju procedūros gražina TRUE arba FALSE reikšmę priklausomai nuo situacijos, ar komunikacija yra užbaigta. Pastarasis atvejis naudingas situacijose, kai mes norime sužinoti, ar komunikacija užbaigta, bet rezultatai šiuo metu ne reikalingi.

1.1.2.2. MPI kolektyvinė komunikacija

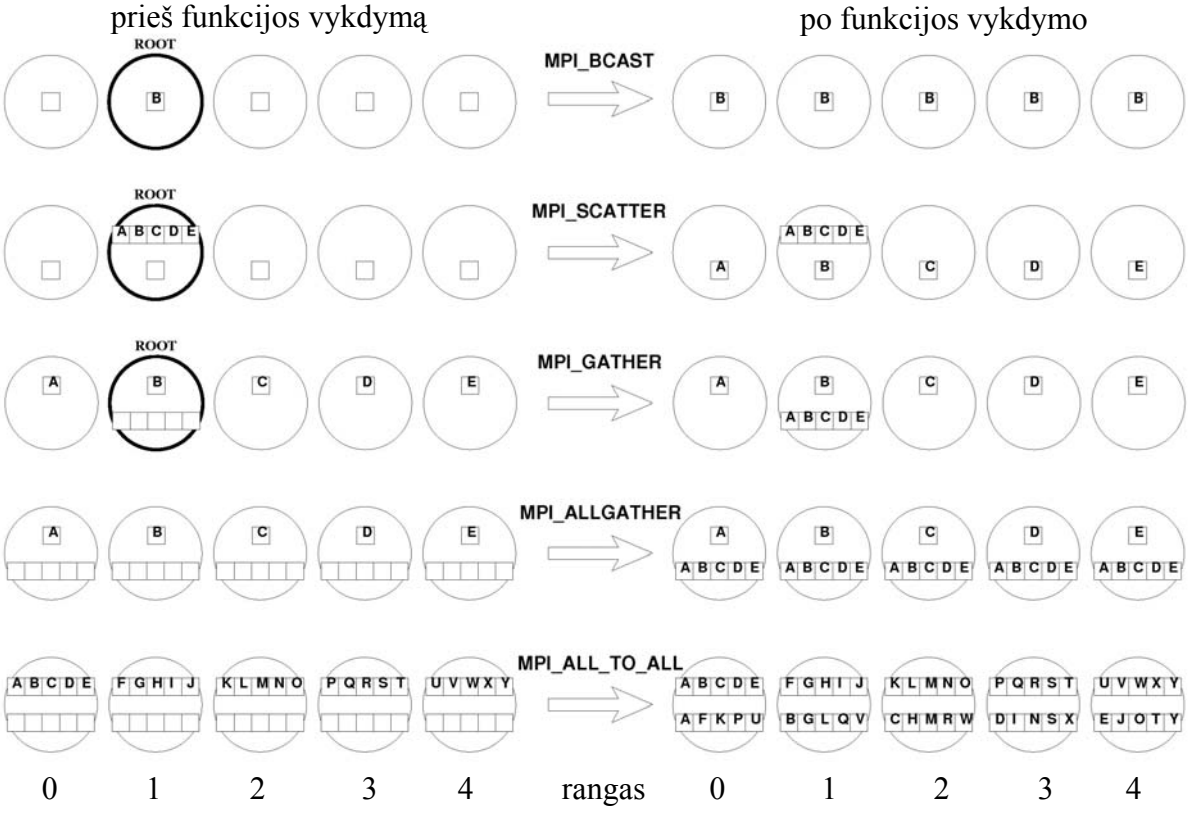
Kolektyvinė komunikacija perduoda duomenis tarp visų komunikatoriaus apibrėžtų procesų. [10] Kolektyvinėje komunikacijoje MPI teikia procedūrų įvairovę duomenų paskirstymui, perskirstymui ir surinkimui. Kad atlikti kolektyvinę komunikaciją, procesų aibei komunikatoriuje turi būti sukurtas naujas komunikatorius. Kolektyvinės komunikacijos savybės [8]:

- kolektyvinė komunikacija netrukdo tiesioginei komunikacijai. Bendradarbiavimas galimas, kai komunikatorius turi du vidinius komunikatorius, kurių vienas skirtas tiesioginei komunikacijai, o kitas – kolektyvinei.
- užbaigimas reiškia, kad buferis gali būti naudojamas toliau arba naudojamas pakartotinai;
- visi procesai komunikatoriuje turi iškviesti kolektyvinę komunikaciją. Tačiau kai kurie procedūriniai argumentai yra neprasmingi kai kuriems procesams ir gali būti apibrėžti kaip “fiktyvi“ reikšmė.
- panašumai su tiesiogine komunikacija:
    - pranešimas yra vieno konkretaus duomenų tipo masyvas.
    - siuntimo ir gavimo duomenų tipas turi sutapti.
- skirtumai:
    - siuntimo pranešimas turi užpildyti apibrėžtą gavimo buferį.

MPI pateikia šias kolektyvinės komunikacijos funkcijas:

- duomenų perdavimas (angl. broadcast) funkcijos pagalba iš vieno proceso siunčiamos pranešim kopijos į visus kitus procesus,
- duomenų paskirstymo (angl. scatter) funkcijos atveju duomenys iš vieno proceso paskirstomi visiems procesams,
- duomenų surinkimo (angl. gathering) funkcijos atveju duomenys siunčiami iš visų procesų į vieną,
- visiško duomenų rinkimo (angl. allgather) atveju duomenų surinkimas taikomas visiems procesams,
- iš visų procesorių duomenų perdavimo visiems procesams (angl. alltoall) funkcijos pagalba visi procesai gauna skirtingus rezultatus,
- barjerinė (angl. barrier) sinchronizacija tarp visų procesų, naudinga procesų sinchronizacijai (duomenų perdavimas čia neliečiamas, MPI_BARRIER blokuoja kvietimo procesą, kol visi kiti grupės nariai yra jį iškvietę). [10]

Broadcast, scatter, gather ir kt. operacijos schematiškai pavaizduotos *8 pav*. Apskritimai reiškia procesus, kvadratai – buferius, o raidės – duomenis.



8 pav. Operacijos [8]

MPI privalumai:

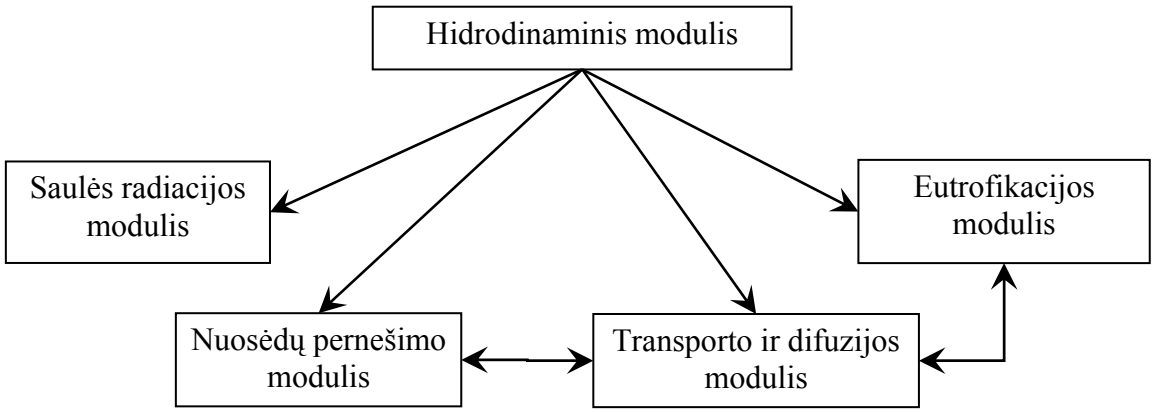
- sujungti į tinklą kompiuteriai gali turėti skirtingas architektūras;
- konstruojant daugiaprocesorines sistemas patiriamos mažesnės išlaidos, nei perkant superkompiuterius.

Trūkumai:

- nėra tiesioginio proceso prieinamumo prie kito proceso atminties;
- informacijos apsikeitimas tarp procesų reikalauja laiko, todėl dideliais kiekiais vykstantis informacijos apsikeitimas neduos efektyvumo;
- reikalinga greita tinklinė įranga, kuri yra brangi.

2. SHYFEM MODELIO APRAŠYMAS

Modelio SHYFEM struktūra pavaizduota (*9 pav.*).



9 pav. SHYFEM

Hidrodinaminis modelis sprendžia hidrodinamines lygtis: [11]

$$\frac{\partial \eta}{\partial t} + \frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} = 0 \text{, kur } U = \int_{-h}^{\eta} u dz \text{ ir } V = \int_{-h}^{\eta} v dz \qquad (1),$$

$$\frac{\partial U}{\partial t} - fV + gH\frac{\partial \eta}{\partial x} + RU + X = 0 \text{, kur } H = h + \eta \qquad (2),$$

$$\frac{\partial V}{\partial t} + fU + gH\frac{\partial \eta}{\partial y} + RV + Y = 0 \qquad (3)$$

kur η – vandens lygis, u, v – horizontalūs greičiai x ir y kryptimis, U ir V – vertikaliai integruoti greičiai, g – gravitacinis pagreitis, f – Koriolio parametras, H – bendras vandens gylis, h – tam tikras vandens gylis, t – laikas, R – trinties koeficientas. X, Y reiškia visas kitas sąvokas kaip, pavyzdžiui, vėjo reikšmė, kuri išreiškiama konstanta, ar netiesinė advekcijos lygtis.

Eutrofikacinis modulis imituoja devynių būsenos kintamųjų dinamiką: ištirpęs deguonis (DO), biocheminis deguonies sunaudojimas (CBOD), fitoplanktono anglis (PHY), amonio azotas (NH3), nitratų ir nitratų azotas (NOX), organinis azotas (ON), organinis fosforas (OP), ortofosfatų fosforas (OPO4) ir zooplanktonas (ZOO) vandenyje ir dugne. Būsenos kintamuosius aprašo tokio pavidalo lygtis:

$$\frac{\partial S}{\partial t} = Q(S) \qquad (4)$$

kur S yra būsenos kintamasis, o Q(S) – viena iš funkcijų, kurios pateiktos *6 lentelė*. [11]

6 lentelė. Eutrofikacinės dalies kintamųjų funkcijos

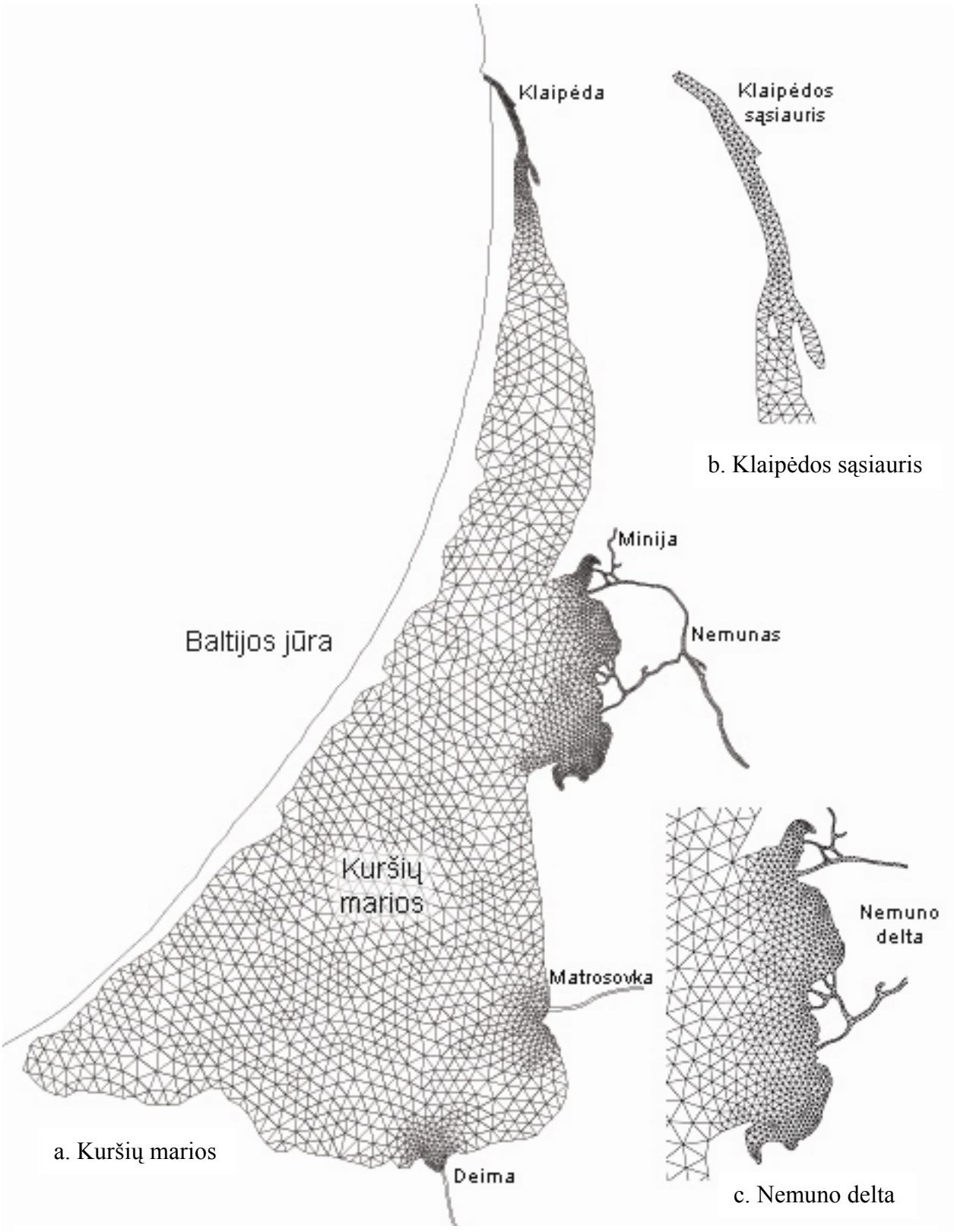
Q(S)	S reikšmė
$Q(PHY) = GPP - DPP - GRZ$	Phytoplankton PHY [mg C/l]
$Q(ZOO) = GZ - DZ$	Zooplankton ZOO [mg C/l]
$Q(NH3) = N_{alg1} + ON1 - N_{alg2} - N1$	Ammonia NH3 [mg N/l]
$Q(NOX) = N1 - NO_{alg} - NIT1$	Nitrate NOX [mg N/l]
$Q(ON) = ON_{alg} - ON1$	Organic nitrogen ON [mg N/l]
$Q(OPO4) = OP_{alg1} + OP1 - OP_{alg2}$	Inorganic phosphorous OPO4 [mg P/l]
$Q(OP) = OP_{alg3} - OP1$	Organic phosphorous OP [mg P/l]
$Q(CBOD) = C1 - OX - NIT2$	Carbonaceous biological oxygen demand CBOD [mg O_2/l]
$Q(DO) = DO1 + DO2 + DO3 - DO4 - N2 - OX - SOD$	Dissolved oxygen DO [mg O_2/l]

Eutrofikacinio modulio būsenos kintamųjų kitimas erdvėje aprašomas advekcijos-difuzijos lygtimi [11]:

$$\frac{\partial \Theta_i}{\partial t} + U\nabla\Theta_i - w_{si}\frac{\partial \Theta_i}{\partial z} = K_h \nabla_H^2 \Theta_i + \frac{\partial}{\partial z}\left[K_v \frac{\partial \Theta_i}{\partial z}\right] + F(\theta, T, I, ...) \quad (5)$$

kur U – greitis, Θ_i – i-tasis būsenos kintamasis, T – vandens temperatūra, I – spinduliavimo lygis, w_{si} – sedimentacijos greitis, K_h ir K_v – horizontalus ir vertikalus turbulentinės difuzijos koeficientai. Daroma prielaida, kad būsenos kintamųjų Θ_i koncentracijos neveikia transporto proceso.

Modelyje diferencialinės lygtys dalinėmis išvestinėmis sprendžiamos baigtinių elementų metodu, paprastos – Oilerio metodu. Lygčių sprendiniai randami kiekviename elemento mazge. Baigtinių elementų metodas patogus tuo, kad leidžia tiksliai aprašyti sudėtingos geometrijos kranto kontūrą, tiksliau pavaizduoti tas zonas, kuriose hidrodinaminis aktyvumas yra didesnis. Dabartinė Kuršių marių gardelė sudarytas iš ~2900 mazgų ir ~4880 trikampinių elementų, kurių dydžiai ir forma yra skirtingi, kad būtų galima gardelės tinklą kuo geriau pritaikyti prie geometrinių ir topografinių ribų. Kiekvienas trikampinio elemento mazgas turi visų kintamųjų reikšmes. [11] *10 pav.* parodytas sudarytas gardelės tinklas: a) visose Kuršių mariose, b) šiaurinėje Kuršių marių dalyje bei c) ties Nemuno delta. Kaip matyti, pastarosiose zonose gardelės tinklas yra žymiai tankesnis, nei likusioje kuršių marių dalyje ir tai leidžia tiksliau aprašyti šiose zonose vykstančius hidrodinaminius procesus.



10 pav. Kuršių marių gardelė

3. LYGIAGREČIŲ SKAIČIAVIMŲ REALIZACIJA SHYFEM MODELIO PROGRAMINIAME KODE

SHYFEM modelis realizuotas Fortran 77 kalba. Buvo nudojamas Intel Fortan kompiliatorius.

Skaitiniai eksperimentai parodė, kad didžiausią laiko dalį yra vykdomas eutrofikacinis modulis, t.y. advekcijos-difuzijos lygties (5) sprendimas. Dėl vykdymo laiko pailgėjimo beveik 3 kartus eutrofikacinė dalis buvo pasirinkta kaip lygiagretiems skaičiavimams tinkanti vieta SHYFEM programos kode. Tai ne vienintelė potenciali vieta lygiagretinimui, nes programos apimtis yra gana didelė. Kitame SHYFEM programos kodo optimizavimo etape numatoma taikyti lygiagrečius skaičiavimus nuosėdų pernešimo modulyje.

Šiuo metu SHYFEM programiniam kodui yra pritaikytos OpenMP ir MPI technologijos. Eksperimentiniai skaičiavimai buvo atlikti 252 dienų periodui (2000.03.15 – 2000.11.22).

3.1. OpenMP pritaikymas

Kompiliuojant programą OpenMP režimu, kompiliatorius turi būti iškviečiamas su –openmp ir -threads opcija, kuri leidžia su OpenMP direktyvomis atlikti skaičiavimus lygiagrečiai, jei reikia nurodomas optimizavimo lygis, kad programa būtų greičiau įvykdyta: –O2 opcija leidžia blokinio lygmens optimizaciją, –O3 opcija prideda funkcinio lygmens globalinę optimizaciją. SHYFEM make failo fragmentas:

```
F77   = ifort
FFLAGS_NOOPT = -threads -openmp -cpp
FFLAGS = $(FFLAGS_NOOPT) -O2 -assume buffered_io

LFLAGS_NOOPT = -threads –openmp
LFLAGS = -O3 $(LFLAGS_NOOPT) -assume buffered_io
```

OpenMP fragmentas programoje:

```
c     ------------------------------------------------------------------
c     advection and diffusion
c     ------------------------------------------------------------------
!$OMP PARALLEL PRIVATE(i)
!$OMP DO SCHEDULE(DYNAMIC)
```

```
      do i=1,nstate
          call scal3sh(what,e(1,1,i),nlvdim,eb(1,1,i),
     +            rkpar,difhv,difv,difmol)
          call tsmass (e(1,1,i),1,nlvdim,tstot(i)) !mass control
      end do

!$OMP END DO NOWAIT
!$OMP END PARALLEL
```

kur i – eutrofikacinio modulio i-tasis būsenos kintamasis yra privatus kiekvienoje gijoje. Turima įranga leidžia modelį paleisti naudojant 2 gijas.

3.2. MPI pritaikymas

Mūsų atveju MPI realizuojamas su MPICH. SHYFEM programos make failo fragmentas:

```
F77   = /opt/mpich/intel/bin/mpif77
FFLAGS_NOOPT =-assume buffered_io
FFLAGS = $(FFLAGS_NOOPT) -O3

LFLAGS_NOOPT = -assume buffered_io
LFLAGS =  -O3 $(LFLAGS_NOOPT)
```

Modelio pagrindinės programos fragmentas MPI atveju:

```
program ht
      include 'param.h'
      include 'mpif.h'
      integer myid, numprocs, ierr

      call MPI_INIT( ierr )
      call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
      call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )

.............

      call MPI_BCAST(it,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
      call MPI_BCAST(itend,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
.............
```

```
      call MPI_BCAST(nlv,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
      call MPI_BCAST(linkv,nlidim,MPI_integer,0,MPI_COMM_WORLD,ierr)

c%%%%%%%%%%%%%%%%%%%%%%%%% time loop %%%%%%%%%%%%%%%%%%%%%%%%%%%%
      do while( it .lt. itend )

       if (myid.eq.0) then
         call set_timestep
         call dobefor3d
         call sp111(2)              !boundary conditions
         call sp259f                !hydro
         call tstvol(saux1,saux2,saux3,saux4,v1v)
         call tstvol1(v1v)
         call conz3sh               !transport/diffusion
         call barocl(1)             !baroclinic contribution
         call subwaves(it)          !wave model
         call sedi(it,idt)          !sediment transport
       endif

............

       iserial=0

       call MPI_BCAST(it,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
       call MPI_BCAST(wdifhv,neldim9,MPI_REAL,0,MPI_COMM_WORLD,ierr)
       ibnnbcdim=ibndim*nbcdim
       call MPI_BCAST(bnd,ibnnbcdim,MPI_REAL,0,MPI_COMM_WORLD,ierr)

............

       call bio3d(it,idt)
      end do

      call MPI_FINALIZE(ierr

end
```

Kintamieji nepriklausantys nuo laiko yra inicializuojami už ciklo ribų. Jų reikšmės perduodamos atitinkamose programos vietose. Šiuo metu modelis MPI atveju gali būti paleistas naudojant 1, 3 ar 9 procesus, nes maksimalus eutrofikacijos būsenos kintamųjų skaičius 9. Turima įranga leidžia modelį paleisti naudojant 3 procesus. MPI fragmentas eutrofikacinėje dalyje:

```
c     ------------------------------------------------------------------
c     advection and diffusion
c     ------------------------------------------------------------------
        call MPI_BCAST(rkpar,1,MPI_REAL,0,MPI_COMM_WORLD,ierr)
        call MPI_BCAST(difmol,1,MPI_REAL,0,MPI_COMM_WORLD,ierr)

       numpiece=nstate/numprocs
       ipiece=numpiece*nkndim

        call MPI_SCATTER(e,ipiece,MPI_REAL,epiece,ipiece,
     + MPI_REAL,0,MPI_COMM_WORLD,ierr)
        call MPI_SCATTER(eb,ipiece,MPI_REAL,ebpiece,ipiece,
     + MPI_REAL,0,MPI_COMM_WORLD,ierr)

       if((numprocs.gt.nstate).or.(nstate.ne.(nstate/numprocs)*numprocs)) then
        write(6,*) 'BIO3D: Number of processes is
       + incompatible with number of state variables!'
        stop
       endif

       do i = 1,numpiece

          ipos1=nkndim*(i-1)+1
          call scal3sh('lagvebio',epiece(ipos1), nlvdim,ebpiece(ipos1),
        + rkpar,difhv,difv,difmol)
       end do

       call MPI_GATHER(epiece,ipiece,MPI_REAL,e,ipiece,MPI_REAL,0,
     + MPI_COMM_WORLD,ierr)

       call MPI_BARRIER(MPI_COMM_WORLD,ierr)
```

Eutrofikacinėje dalyje MPI atveju kintamieji padalinami visiems procesams ir iš visų procesų surenkami į vieną, pasibaigus adveksijos-difuzijos skaičiavimams.

4. MODELIO GREITAEIGIŠKUMO ANALIZĖ

Modelio greitaeigiškumui patikrinti, eksperimentai buvo atlikti naudojant IA64 architektūros dviejų mazgų klasterį, kurio pagrindiniai parametrai surašyti *7 lentelė*.

7 lentelė. IA64 klasterio parametrai

Mazgo vardas	bgm-corpi	compute-0-0
Operacinė sistema	Roks Linux 2.4.21-20.EL	
Procesorių skaičius	2	2
Procesorių tipas	Intel(R) Itanium(R) II	
Kiekvieno procesoriaus greitis, GHz	1,36	1,56
Kiekvieno procesoriaus atmintis, GB	1,94	1,94
Mazgo disko dydis, GB	69,248	71,332

Efektyvumui nustatyti gali būti pasirinktos charakteristikos [3]:

- Spartinimo koeficientas $S_p = \frac{T_0}{T_p}$, įvertinantis pagreitėjimą, kurį pasiekiame spręsdami uždavinį lygiagrečiuoju algoritmu, naudodami p procesorių skaičių (T_0 – greičiausio nuosekliojo algoritmo vykdymo laikas, T_p – lygiagrečiojo algoritmo naudojant p procesorius vykdymo laikas);
- Efektyvumo koeficientas $E_p = \frac{S_p}{p}$, parodantis, kokią dalį procesorių pajėgumo pasitelkėme spręsdami uždavinį lygiagrečiuoju algoritmu.
- Amdahl'o dėsnis naudojamas surasti maksimalų laukiamą paggreitėjimą. Jeigu F yra nuoseklaus skaičiavimo dalis ir 1-F dalis, kuri gali būti lygiagretinama, N – procesorių skaičius, tai maksimalų spartinimo koeficientą galima apskaičiuoti pagal formulę [12]

$$S_p \max = \frac{1}{F + \frac{1-F}{N}}$$

Skaičiavimus atliekant OpenMP atveju naudojami 2 procesoriai, MPI atveju – 3 procesoriai. 252 dienų prognozės vykdymo laiko rezultatai pateikti *8 lentelė*.

8 lentelė. Lygiagretaus skaičiavimo spartinimo ir efektyvumo koeficientai

Programos vykdymas	Procesorių skaičius, p	Skaičiavimo laikas, s	Paros skaičiavimo laikas, min	Pagreitėjimas, S_p	Efektyvumas, E_p
Nuosekliai	1	18420	1,22	–	–
Pritaikius OpenMP	2	15240	1,00	1,21	0,6
Pritaikius MPI	3	13200	0,87	1,4	0,5

Pritaikę Amdahl'o dėsnį, galėjome įvertinti, kiek maksimaliai galima sumažinti programos vykdymo laiką, kai žinomas laikas, tenkantis nuosekliai vykdomai algoritmo daliai. Šiuo atveju F laikėme programos vykdymo be eutrofikacinės dalies ir su eutrofikacine dalimi santykį, darydami prielaidą, kad eutrofikacinės dalies vykdymo laikas apytiksliai yra lygus advekcijos-difuzijos lygties skaičiavimo laikui. Modelio programa buvo įvykdyta per 1 val. 45 min. (6300 s), kai įvedimo faile eutrofikacinės dalies parametras buvo prilygintas nuliui, ir per 5 val. (18000 s), kai įvedimo faile eutrofikacinės dalies parametras prilyginamas vienetui. Gavus rezultatus, įvertiname F = 0,35. Maksimalus procesorių skaičius N negali viršyti 9, nes eutrofikacinėje dalyje tik 9 kintamieji. 5 lygties skaičiavimai, esant skirtingam procesorių skaičiui, pateikti *9* ***lentelė.*** *lentelėje*.

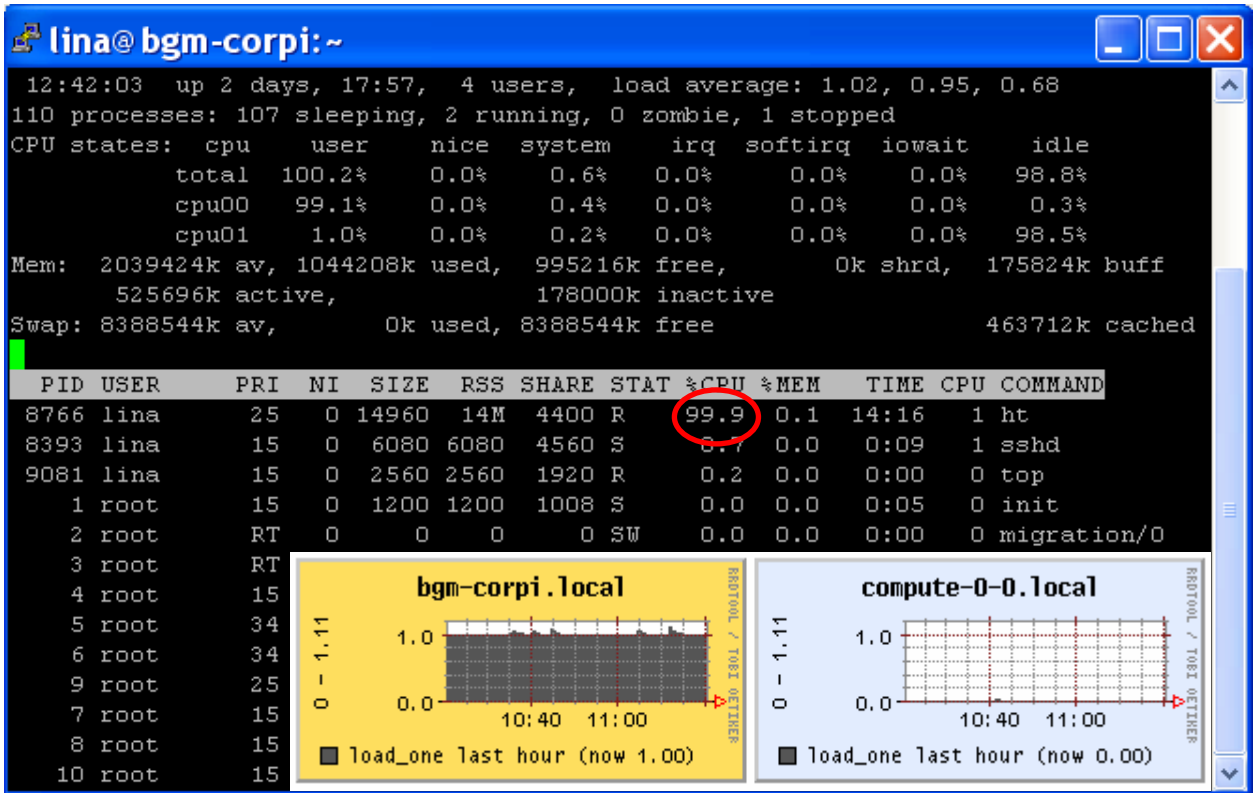
9 lentelė. Amdahl dėsnio skaičiavimai, esant skirtingam procesorių skaičiui

Eil. Nr.	Procesorių skaičius, (N)	Nuoseklaus skaičiavimo dalis, (F)	Galima dalis lygiagretinimui, (1-F)	$\frac{1}{F+\frac{1-F}{N}}$
1.	2	0,35	0,65	1,48
2.	3			1,76
3.	4			1,95
4.	9			2,38

Palyginus *8* ***lentelė.*** ir *9* ***lentelė.*** *lentelių* rezultatus, matome, kad tiek OpenMP, tiek MPI atvejais yra nepasiektas maksimalus vykdymo laiko paspartėjimas. OpenMP SHYFEM modeliui pritaikytas geriau nei MPI, nes efektyvumo koeficientas yra didesnis ir eksperimento rezultatai yra arčiau teorinių, tačiau technologijas lyginant atsižvelgus į pagreitėjimo koeficientą, MPI atveju modelis įvykdomas greičiau.

Palyginus gautus spartos koeficientus iš *8 lentelė.*, matome, kad pagreitėjimo laikas tarp programos paleidimo naudojant du ar tris procesorius skiriasi ne daug. Galimos priežastys: ne pilnai išnaudojami procesoriai, nepasiekiamas maksimalus pagreitėjimas. Antroji hipotezė buvo patvirtinta palyginus eksperimentinius (*8 lentelė.*) ir teorinius (*9 lentelė.*) rezultatus.

Pirmąją hipotezę patikrinome pasinaudoję Linux komanda *top*, kuri pastoviai atnaujina naudojamų procesorių išnaudojimą sistemoje, ir http://bgm-corpi.ku.lt/ganglia/ puslapiu, kuriame pateikiami procesorių užimtumo grafikai. Pirmiausia programa buvo paleista nuosekliu režimu, t.y. naudojant vieną procesorių, kad vėliau galėtumėm palyginti procesorių užimtumą, paleidus programą lygiagrečiu režimu. Klasterio mazgų išnaudojimo procentai pateikti *10 lentelė. 11 pav.* pateiktas klasterio apkrovimas modelio nuosekliu režimu vykdymo metu: *bgm-corpi* mazgo apkraunamas tik 50 %, nes programa buvo paleista naudojant tik vieną procesorių, kuris išnaudojamas 100 %, o *compute-0-0* mazgas nenaudojamas visai.

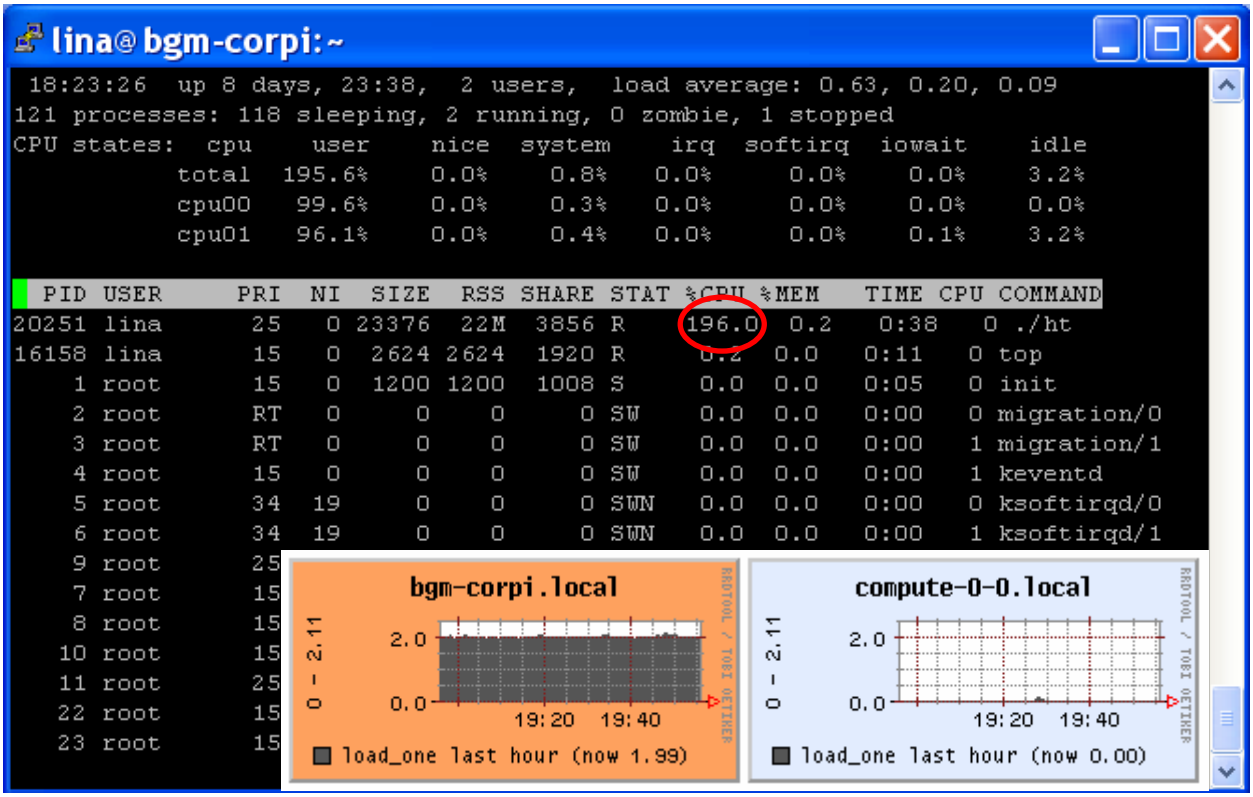


11 pav. Klasterio užimtumas modelio vykdymo nuosekliu režimu metu

10 lentelė. Mazgų užkrovimas programos vykdymo metu (http://bgm-corpi.ku.lt/ganglia/)

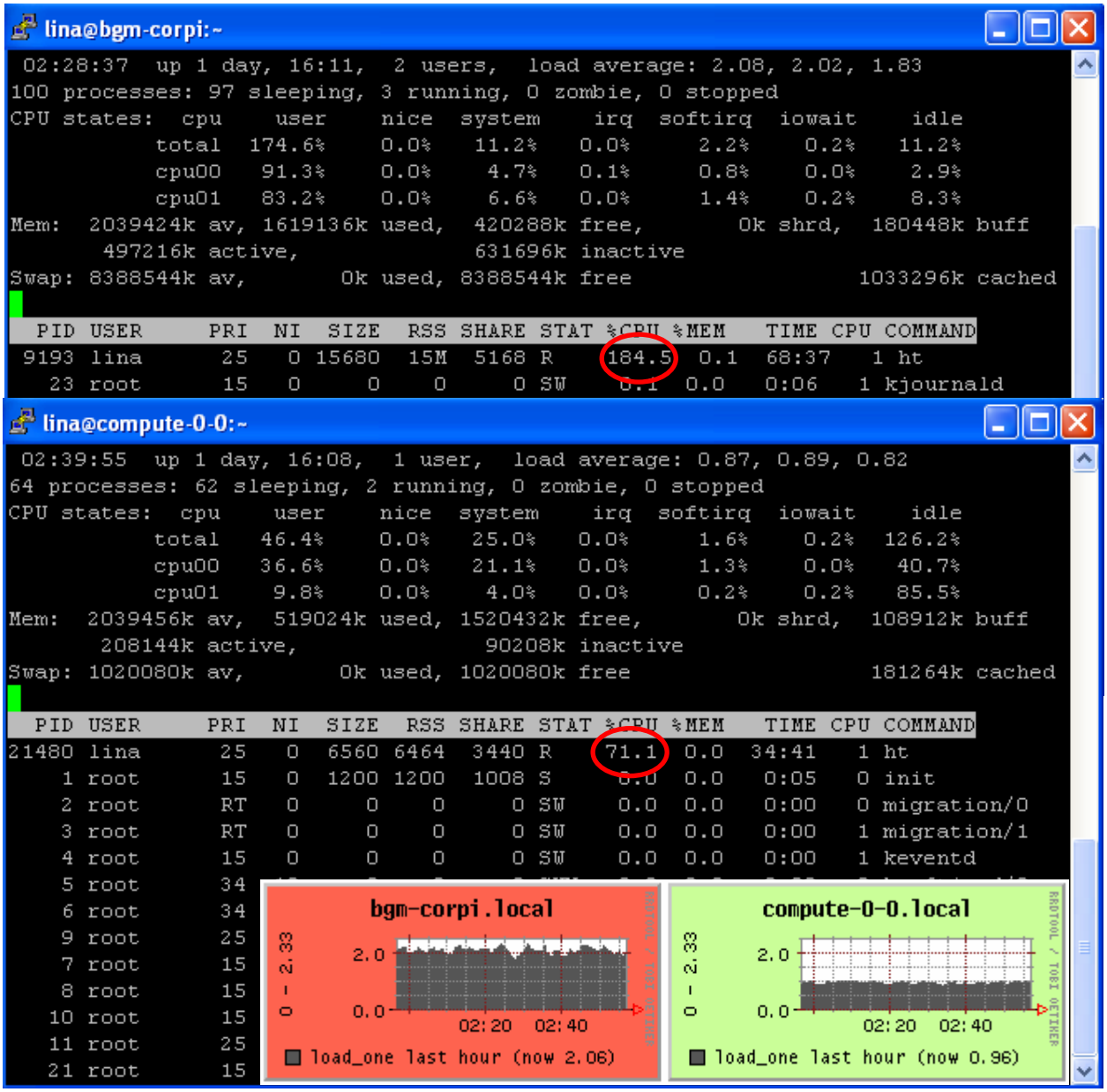
Spalva, reiškianti mazgo apkrovimą	$\frac{mazgo\_apkrovimas\_per\_1\min.}{CPU\_nr.} x100\%$
Raudona	virš 100 %
Oranžinė	75 – 100 %
Geltona	50 – 74 %
Žalia	25 – 49 %
Mėlyna	0 – 24%

OpenMP atveju, *12 pav.*, *bgm-corpi* mazgas apkraunamas 100 %, nes programa paleidžiama naudojant abu procesorius, iš kurių vienas išnaudojamas 100 % ir kitas 96 %, o *compute-0-0* mazgas nenaudojamas visai kaip ir nuoseklaus režimo atveju.



12 pav. Klasterio užimtumas modelio vykdymo metu naudojant 2 procesorius

MPI atveju, *13 pav.*, *bgm-corpi* mazgas apkraunamas 100 %, o *compute-0-0* mazgas – iki 50 %. Programa vykdoma naudojant tris procesorius: du *bgm-corpi*, iš kurių vienas išnaudojamas apie 96 % ir kitas 99 %, ir vieną *compute-0-0* procesorių, kuris išnaudojamas apie 71 %, iš kurių virš 20 % naudoja sistema. Taigi paleidus programą naudojant tris procesorius, trečiasis procesorius išnaudojamas ne pilnai.



13 pav. Klasterio užimtumas modelio vykdymo metu naudojant 3 procesorius

IŠVADOS

Darbe nagrinėjamas SHYFEM modelio programinio kodo lygiagretinimo galimybės. Lygiagrečių skaičiavimų technologijų taikymas pasirinktas difuzijos-advekcijos lygčiai spręsti. Lygiagretūs skaičiavimai atlikti dviem metodais: OpenMP ir MP. Atlikus greitaeigiškumo analizę, galime daryti išvadas:

1. MPI atveju modelio greitaeigiškumo rezultatai (1,4) yra geresni nei OpenMP atveju (1,2);
2. remiantis Amdahl'o dėsniu, OpenMP atveju modelio vykdymo paspartėjimas (realus – 1,21; maksimalus – 1,48) yra arčiau maksimalaus paspartėjimo nei MPI atveju (realus – 1,4; maksimalus – 1,76);
3. teoriniai skaičiavimai rodo, kad padidinus procesorių skaičių iki 9, modelio vykdymo laikas turėtų sutrumpėti 0,43 karto (4 procesorių pagreitėjimas – 1,95, 9 procesorių – 2,38).

LITERATŪRA

1. VGTU klasteris Vilkas, vartotojo vadovas, [interaktyvus], [žiūrėta 2004 02 28], http://vilkas.vtu.lt/Vartotojo_vadovas.htm
2. VGTU skaičiavimo centras, [interaktyvus], [žiūrėta 2005 02 28], http://www.vgtu.lt/sc/?id=23
3. Gintautas Dzemyda, Virginijus Marcinkevičius, Olga Kurasova. MPI programų paketo taikymas lygiagrečiajam vizualizavimui. INFORMACIJOS MOKSLAI, Vilniaus universiteto leidykla, 2003, 26 tomas, ISSN 1392-0561
4. Parallel and Distributed Data Intensive Computing, [interaktyvus], paskutinį kartą keistas 2002 12 13, [žiūrėta 2005 03 02] http://www.cs.umd.edu/class/fall2002/cmsc818s/Lectures/jeff-pvm-mpi.pdf
5. OpenMP Architecture Review Board, [interaktyvus], [žiūrėta 2005 03 03], http://www.openmp.org/drupal/node/view/11
6. http://www.sdsc.edu/~allans/cs260/lectures/OpenMP.ppt [žiūrėta 2005 03 17]
7. Gabriel Mateescu. Writing and Tuning OpenMP Programs on Distributed Shared Memory Machines. [interaktyvus], [žiūrėta 2005 03 17] http://www.sao.nrc.ca/~gabriel/openmp/openmp.ppt
8. Neil MacDonald, Elspeth Minty, Joel Malard, Tim Harding, Simon Brown, Mario Antonioletti. Writing Message-Passing Parallel Programs with MPI, [interaktyvus], The University of Edinburgh, [žiūrėta 2005 02 24] http://www.epcc.ed.ac.uk/computing/training/document_archive/mpi-course/mpi-course.pdf
9. OpenMP, [interaktyvus], paskutinį kartą keistas 2005 06 04, [žiūrėta 2005 02 25] http://www.llnl.gov/computing/tutorials/openMP/
10. Jack J. Dongarra, Steve W. Otto, Marc Snir, and David Walker. A message passing standard for MPP and workstations, COMMUNICATIONS OF THE ACM, July 1996, Vol. 39, No. 7
11. Georg Umgiesser, Donata Melaku Canu, Cosimo Solidoro, Robert Ambrose. A finite element ecological model: a first application to the Venine Lagoon. Environmental Modelling & Software, 2003, No. 18, pp. 131–145, 10.1016/S1364-8152(02)00056-7
12. Enciklopedija, [interaktyvus], [žiūrėta 2005 06 03], http://en.wikipedia.org/wiki/Amdahl's_law