МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

МЕТОДИЧНІ ВКАЗІВКИ
ЩОДО ВИКОНАННЯ САМОСТІЙНИХ РОБІТ
З НАВЧАЛЬНОЇ ДИСЦИПЛІНИ
«ПАРАЛЕЛЬНІ ТА РОЗПОДІЛЕНІ ОБЧИСЛЕННЯ»
ДЛЯ СТУДЕНТІВ ДЕННОЇ ФОРМИ НАВЧАННЯ
ЗІ СПЕЦІАЛЬНОСТІ 123 – «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»

КРЕМЕНЧУК 2018

Методичні вказівки щодо виконання самостійних робіт з навчальної дисципліни «Паралельні та розподілені обчислення» для студентів денної форми навчання зі спеціальності 123 – «Комп'ютерна інженерія»

Укладач к. т. н., доц. О. Г. Славко

Рецензент к. т. н., доц. В. М. Сидоренко

Кафедра комп'ютерних та інформаційних систем

Затверджено методичною радою КрНУ імені Михайла Остроградського
Протокол № 11 від 9 липня 2018 р.

Голова методичної ради _____ проф. В. В. Костін

ЗМІСТ

ВСТУП

Цей навчальний курс є логічним продовженням навчальних курсів «Комп'ютерні системи» і «Програмування». Предметом вивчення навчальної дисципліни є системні рішення, що покладені в основу побудови паралельних алгоритмів, орієнтованих для реалізації на багатопроцесорних обчислювальних системах, а також методи та підходи до організації паралельних обчислень і способи організації обчислювальних кластерів і архітектури багатопроцесорних систем.

У межах навчального курсу розглядаються питання дослідження особливостей функціонування сучасних комп'ютерних систем, орієнтованих на організацію та реалізацію паралельних обчислень. Студенти ознайомлюються з основами розробки ефективних паралельних алгоритмів, використання паралелізму програм, набувають практичних навичок з використання технологій паралельного програмування, використовуючи при цьому мову C++ та бібліотеку MPI (Message passing interface), що надає низькорівневий, але водночас надзвичайно зручний інтерфейс програмування для мережного кластера та базується на ідеології обміну повідомленнями між паралельними процесами, ознайомлюються з інтегрованим середовищем розробки паралельних програм AdaGIDE.

Паралельні обчислення – це такий спосіб організації комп'ютерних обчислень, за якого програми розробляються як набір взаємодіючих обчислювальних процесів, що працюють паралельно (одночасно). Термін охоплює сукупність питань паралелізму в програмуванні, а також створення ефективних апаратних реалізацій. Теорія паралельних обчислень становить розділ прикладної теорії алгоритмів.

Основна складність у проектуванні паралельних програм – це забезпечення правильної послідовності взаємодій між різними обчислювальними процесами, а також координація ресурсів, розподілених між процесами.

Мета та завдання навчальної дисципліни: набуття студентами загальних теоретичних і практичних знань у галузі паралельної обробки інформації, а також з питань, пов'язаних з аналізом методів і алгоритмів паралельної обробки інформації, дослідженням моделей і мов паралельних обчислень, ознайомлення з методами та алгоритмами паралельних обчислень, оволодіння навичками організації паралельної та розподіленої обробки інформації в суперкомп'ютерах і паралельних системах.

Місце навчальної дисципліни у навчальному процесі: навчальний курс «Паралельні та розподілені обчислення» тісно пов'язаний з такими навчальними дисциплінами, як «Комп'ютерні системи», «Мережні інформаційні технології», «Дискретна математика», «Програмування» та іншими спеціальними дисциплінами навчального плану.

У результаті вивчення навчальної дисципліни студент повинен

знати:

– теоретичні та практичні аспекти паралельної обробки інформації;

– особливості організації обчислювальних процесів у паралельних обчислювальних системах;

– структури та принципи побудови програмного забезпечення розподілених систем обробки інформації;

– моделі системи планування обчислювальних процесів;

уміти:

– орієнтуватись у класах сучасних суперкомп'ютерів і моделях паралельної обробки інформації;

– вибирати модель, метод та алгоритм, найбільш адекватний для ефективної паралельної обробки для розв'язання того чи іншого класу задач;

– створювати додатки користувача для паралельної обробки інформації із застосуванням спеціалізованих бібліотек і однієї з мов паралельної обробки інформації.

1 ТЕМИ ТА ПОГОДИННИЙ РОЗКЛАД ЛЕКЦІЙ І САМОСТІЙНОЇ РОБОТИ

Назви змістових модулів і тем	Кількість годин				
	Денна форма				
	Усього	У тому числі			
		л	п	лаб	с.р.
1	2	3	4	5	6
Модуль 1					
Змістовий модуль 1 Паралельні та розподілені алгоритми					
Тема 1 Принципи побудови паралельних обчислювальних систем	10	2	-	-	8
Тема 2 Аналіз паралельних обчислень та основні властивості паралельних алгоритмів	10	2	-	-	8
Тема 3 Парадигми паралельних додатків	10	2	-	-	8
Разом за змістовим модулем 1	30	6	-	-	24
Змістовий модуль 2 Програмування в паралельних і розподілених системах					
Тема 4 Розробка паралельних програм	16	2	2	2	10
Тема 5 Мови паралельного програмування	16	2	2	2	10
Тема 6 Особливості використання стандартів OpenMP, MPI і CUDA	20	2	4	4	10
Тема 7 Бібліотечні функції для паралельного програмування	20	2	4	4	10
Усього за змістовим модулем 2	72	8	12	12	40
Змістовий модуль 3 Підходи до організації паралельного програмування					
Тема 8 Задача розміщення	10	2	-	2	6
Тема 9 Оптимізація обміну даними	8	2	-	-	6

1	2	3	4	5	6
Тема 10 Засоби опису паралельних процесів	10	2	-	2	6
Усього за змістовим модулем 3	28	6	-	4	18
Змістовий модуль 4 Технології взаємодії та синхронізації процесів					
Тема 11 Засоби взаємодії та синхронізації процесів	12	2	-	2	8
Тема 12 Задача взаємного виключення	10	2	-	2	6
Усього за змістовим модулем 4	22	4	-	4	14
ІНДЗ (КР, РГ, к/р)	6	-	-	-	6
Семестровий контроль (залік)	4	-	-	-	4
Усього годин	162	24	12	20	106

2 ПЕРЕЛІК САМОСТІЙНИХ РОБІТ

Самостійна робота № 1

Тема. Вступ до стандарту MPI

Мета: вивчення базової структури MPI-програм і способу їх запуску на кластері.

Короткі теоретичні відомості

Базова структура MPI-програм передбачає обов’язкове застосування функцій:

– MPI_Init;

– MPI_Comm_size;

– MPI_Comm_rank;

– MPI_Finalize.

Їх використання наведено в прикладі нижче.

```
#include "mpi.h"
```

```
#include <stdio.h>
int main(argc,argv)
int argc;
char *argv[]; {
int  numtasks, rank, rc;
rc = MPI_Init(&argc,&argv);
if (rc != MPI_SUCCESS) {
  printf ("Error starting MPI program. Terminating.\n");
  MPI_Abort(MPI_COMM_WORLD, rc);
  }
MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
printf ("Number of tasks= %d My rank= %d\n", numtasks,rank);
MPI_Finalize();
}
```

Компіляція MPI-програм здійснюється такими утилітами:

– **mpicc** (для програм, написаних мовою C);

– **mpiCC** (для програм, написаних мовою C++).

Ці утиліти є надбудовами над відповідними компіляторами, установленими в рамках операційної системи.

Приклад запуску MPI-програми на компіляцію:

```
mpiCC myprog.cpp –o myprog
```

Запуск відтрансльованої програми проводиться за допомогою команди виду:

```
mpirun [параметри mpirun] <ім'я програми> [вхідні параметри програми]
```

Основними параметрами **mpirun** є:

- **h | -help** – показує можливі аргументи і параметри команди **mpirun;**
- **maxtime** – задає максимальний час обчислень програми (у хвилинах);
- **np <кількість процесорів>** – задає кількість процесорів, на яких запускається програма.

Приклад запуску:

```
mpirun –np 4 –maxtime 5 myprog
```

Порядок виконання самостійної роботи

Задача 1. Відкомпілювати і виконати програму, наведену вище, яка задає базову структуру MPI-програм. Запустити її на кількості процесорів від 1 до 4.

Задача 2. Написати й налагодити MPI-програму, у якій кореневий процес (процес з рангом 0) видає на консоль кількість запущених процесів, а також повідомлення «Root process: Hello, world», а всі інші процеси видають повідомлення «Slave process: my rank is xxxx», де xxx – це ранг відповідного процесу.

Виконати запуски програми на різній кількості процесорів.

Зміст звіту

1. Назва та мета роботи.
2. Лістинги розроблених програм і результати їх роботи.
3. Відповіді на питання для самоперевірки.

Питання для самоперевірки

1. Назвіть основні види архітектур паралельних обчислювальних систем.
2. У чому полягає специфіка векторно-конвеєрних архітектур?
3. Назвіть основні особливості паралельних обчислювальних систем із загальною пам’яттю.
4. Яка специфіка паралельних обчислювальних систем з розподіленою пам’яттю?
5. Які основні критерії класифікації Флінна? Назвіть основні групи обчислювальних систем, що відповідають цій класифікації.
6. Які види обчислювальних систем належать до SISD, SIMD, MISD і MIMD архітектур?

Література: [2, с. 24–36; 6, с. 42–51; 7, с. 112–128; 14, с. 79–88].

Самостійна робота № 2

Тема. Методи передавання даних типу «точка–точка» в MPI

Мета: вивчити призначення та застосування функцій MPI_Send і MPI_Recv і їх різновидів для обміну даними між MPI-процесами.

Короткі теоретичні відомості

Частіше використовуваними блокувальними функціями передавання даних типу «точка–точка» є функції `MPI_Send` і `MPI_Recv`.

Неблокувальною функцією приймання–передавання називають таку, яка завжди спочатку повертає керування в програму, а потім починає займатися передачею даних. Повернення з такої функції указує лише на те, що наша заявка на приймання або передавання врахована системою та коли-небудь буде виконана.

Для отримання дійсного статусу процесу передавання є ряд функцій `MPI_Wait*()` і `MPI_Test*()`. Неблокувальне передавання зазвичай передбачає дві дії: ініціалізацію передавання (приймання) та перевірку факту її завершення. Для зазначення неблокувального характеру функції до неї приписують префікс «I», наприклад, `MPI_Isend()` – неблокувальний варіант функції `MPI_Send()`.

Неблокувальна функція приймання (`MPI_IRecv()`) повідомляє систему, що можна почати (або очікувати) приймання у фоновому режимі. Чи завершено це приймання, необхідно перевірити додатково.

Блокувальна функція не повертає керування, поки не виконає якусь суттєву та цілком визначену частину приймання або передавання. Яку саме – залежить від режиму роботи викликаної функції.

Існує чотири режими:

1) синхронне передавання (`MPI_Ssend()`): повернення з функції передавання означає, що дані вже почали передаватися одержувачу – безпосередньо з тієї ділянки в пам’яті, де вони лежать. Якщо це блокувальний виклик, то після повернення з нього відомо, що буфер передавання вже можна

використовувати;

2) буферизоване передавання (`MPI_Bsend()`): повернення з функції передавання відбувається, коли передане повідомлення взято в системний або користувацький буфер, буфер передавання також готовий до повторного використання після повернення з функції;

3) передавання за готовністю (`MPI_Rsend()`) схоже на синхронне передавання, але перевірка готовності одержувача приймати дані не виконується, тобто, якщо в момент ініціалізації передавання одержувач не готовий, то відбувається помилка. Передавання за готовністю в деякому розумінні реалізує *рандеву*, тобто передавання відбудеться тільки в тому випадку, якщо одержувач її очікує;

4) автоматичний режим передавання (`MPI_Send()`) припускає, що система сама робить вибір на користь синхронного або буферизує передавання на підставі розміру переданих даних.

Блокувальна функція прийому існує одна для всіх режимів (`MPI_Recv()`), і повернення з неї означає, що дані прийняті.

Порядок виконання самостійної роботи

Задача 1. Відтранслювати і запустити на виконання з параметром -np 2 програму, показану нижче, у якій процес з номером 0 «пінгує» (посилає тестове повідомлення, у даному випадку довжини 1) процес з номером 1 і чекає від нього відповідного повідомлення.

```
#include "mpi.h"
#include <stdio.h>
int main(argc,argv)
int argc;
char *argv[];  {
int numtasks, rank, dest, source, rc, count, tag=1;
char inmsg, outmsg='x';
MPI_Status Stat;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
if (rank == 0) {
  dest = 1;
  source = 1;
```

```
  rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
  rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD,
&Stat);   }
else if (rank == 1) {
  dest = 0;
  source = 0;
  rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD,
&Stat);
  rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
rc = MPI_Get_count(&Stat, MPI_CHAR, &count);
printf("Task %d: Received %d char(s) from task %d with tag %d \n",
       rank, count, Stat.MPI_SOURCE, Stat.MPI_TAG);
MPI_Finalize(); }
```

Задача 2. Запустити програму з параметром -np N, де $N > 2$, і пояснити поведінку MPI-процесів з рангом > 2.

Задача 3. Змінити програму із завдання 1 так, щоб процес з номером 0 «пінгував» усі процеси в групі (окрім самого себе), посилаючи їм однобайтові повідомлення та приймаючи від них такі самі відповідні повідомлення.

Задача 4. Відтранслювати і запустити на виконання програму, показану нижче, у якій між сусідніми процесами, об’єднаними в коло, відбувається обмін повідомленнями, що містять ранг процесу, який посилає.

Вивчити призначення та способи застосування команд `MPI_Isend`, `MPI_Irecv`, `MPI_Waitall`.

Задача 5. Пояснити поведінку програми під час її запуску з параметром –np 1.

```
#include "mpi.h"
#include <stdio.h>
int main(argc,argv)
int argc;
char *argv[];  {
int numtasks, rank, next, prev, buf[2], tag1=1, tag2=2;
MPI_Request reqs[4];
MPI_Status stats[4];
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
prev = rank-1;
next = rank+1;
if (rank == 0)  prev = numtasks - 1;
if (rank == (numtasks - 1))  next = 0;
MPI_Irecv(&buf[0], 1, MPI_INT, prev, tag1, MPI_COMM_WORLD,
&reqs[0]);
MPI_Irecv(&buf[1], 1, MPI_INT, next, tag2, MPI_COMM_WORLD,
```

```
&reqs[1]);
MPI_Isend(&rank, 1, MPI_INT, prev, tag2, MPI_COMM_WORLD,
&reqs[2]);
MPI_Isend(&rank, 1, MPI_INT, next, tag1, MPI_COMM_WORLD,
&reqs[3]);
  {  do some work  }
MPI_Waitall(4, reqs, stats);
MPI_Finalize(); }
```

Зміст звіту

1. Назва та мета роботи.

2. Лістинги розроблених програм і результати їх роботи.

3. Відповіді на питання для самоперевірки.

Питання для самоперевірки

1. Основні види паралелізму для побудови паралельних програм.

2. У чому полягає специфіка реалізації алгоритму множення матриць для SIMD- і MIMD-архітектур?

3. Яка специфіка методу «портфель завдань»?

4. У чому полягає метод організації потоків за принципом діхотомії?

Література: [3, с. 119–132; 6, с. 45–63; 7, с. 250–264; 11, с. 319–340; 14, с. 225].

Самостійна робота № 3

Тема. Колективні (радіомовні) обміни даними між MPI-процесами

Мета: вивчити призначення та застосування функцій обміну даними MPI_Bcast, MPI_Gather, MPI_Scatter між MPI-процесами.

Короткі теоретичні відомості

Відмінності операцій колективного обміну даними від комунікацій типу «точка–точка» такі:

1) приймають/передають дані одночасно всі процеси групи для вказаного комунікатора;

2) операція колективного обміну виконує одночасно і прийом, і передачу даних, а тому вона має параметри, частина з яких відноситься до прийому даних, а частина – їх передачі;

3) як правило, значення всіх параметрів (за винятком адрес буферів з даними) повинні бути тотожні у всіх процесах;

4) для повідомлень, переданих за допомогою колективного обміну, ідентифікатори (tags) призначаються автоматично; окрім того, повідомлення передаються, насправді, не по вказаному комунікатору, а за допомогою тимчасового комунікатора-дубліката, чим забезпечується ізоляція операцій колективної передачі даних один від одного і від операцій типу «точка–точка».

Порядок виконання самостійної роботи

Завдання 1. Реалізувати розсилання повідомлення від процесу із заданим рангом (цей ранг є вхідним параметром задачі) усім іншим процесам у групі за допомогою функції `MPI_Bcast`. Реалізувати еквівалентний варіант програми, у якому замість функції `MPI_Bcast` використовуються функції синхронної передавання даних `MPI_Send` і `MPI_Recv`.

Завдання 2. Реалізувати посилання повідомлень (наприклад, значень рангу) від всіх процесів у групі, процесу із заданим рангом (цей ранг є вхідним параметром програми) за допомогою функції MPI_Gather.

Реалізувати еквівалентний варіант програми, у якому замість функції `MPI_Gather` використовуються функції синхронної передачі даних `MPI_Send` і `MPI_Recv`.

Завдання 3. Відтранслювати і виконати з параметром -np 4 програму, наведену нижче, у якій функція `MPI_Scatter` використовується для розсилання рядків матриці чисел точкою, що плаває, по окремих процесах.

Доповнити цю програму фрагментом, у якому окремі процеси підсумовують усі елементи отриманого рядка та повертають отримані

результати кореневого процесу за допомогою функції `MPI_Gather`.

```
#include "mpi.h"
#include <stdio.h>
#define SIZE 4
int main(argc,argv);
int argc;
char *argv[];  {
int numtasks, rank, sendcount, recvcount, source;
float sendbuf[SIZE][SIZE] = {  {1.0, 2.0, 3.0, 4.0},
                               {5.0, 6.0, 7.0, 8.0},
                               {9.0, 10.0, 11.0, 12.0},
                               {13.0, 14.0, 15.0, 16.0}  };
float recvbuf[SIZE];
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
if (numtasks == SIZE) {  source = 1;
  sendcount = SIZE;
  recvcount = SIZE;
  MPI_Scatter(sendbuf,sendcount,MPI_FLOAT,recvbuf,recvcount,
            MPI_FLOAT,source,MPI_COMM_WORLD);
  printf("rank= %d  Results: %f %f %f %f\n",rank,recvbuf[0],
         recvbuf[1],recvbuf[2],recvbuf[3]);  }
else
  printf("Must specify %d processors. Terminating.\n",SIZE);
MPI_Finalize(); }
```

Зміст звіту

1. Назва та мета роботи.

2. Результати роботи програми *hello.ads* з папки *Examples*.

3. Лістинги розроблених програм за індивідуальними варіантами та результати їх роботи.

4. Письмові відповіді на питання для самоперевірки.

Питання для самоперевірки

1. Для чого призначена бібліотека Pthread?

2. Навести загальну структуру багатопотокової програми, написаної з використанням бібліотеки Pthread.

3. Як компілюється додаток, що використовує бібліотеку Pthread?

4. Як синхронізуються потоки в бібліотеці Pthread?

5. Назвіть основні синхропримітиви, реалізовані в бібліотеці Pthread.

6. Опишіть формат функції, що забезпечує створення потоку в бібліотеці Pthread.

7. Які основні функції керування потоками у бібліотеці Pthread?

8. Для чого використовуються атрибути потоків? Які функції використовуються для роботи з ними?

9. Що таке м'ютекс? Як м'ютекси використовуються для синхронізації потоків?

Література: [1, с. 500–515; 4, с. 110–119; 11, с. 349–358; 13, с. 97–115].

Самостійна робота № 4

Тема. Колективні операції та їх виконання

Мета: вивчити призначення та застосування функцій для виконання колективних операцій MPI_Reduce, MPI_Allreduce, MPI_Reduce_scatter і MPI_Scan.

Короткі теоретичні відомості

Схема виконання колективних операцій MPI-процесами полягає в тому, що кожен MPI-процес має вектор даних (найчастіше, числовий) з елементами певного типу, наприклад, `MPI_INT`. Функція колективної операції (наприклад, `MPI_Reduce`) виконує деяку операцію, яка задається одним з аргументів цієї функції, над нульовими осередками всіх векторів і над першими осередками.

Функції `MPI_Reduce`, `MPI_Allreduce`, `MPI_Reduce_scatter` і `MPI_Scan`, що виконують і колективні операції, відрізняються один від одного тільки способом розміщення результату операції в MPI-процесах.

Порядок виконання самостійної роботи

Задача 1. Реалізувати програму такого змісту:

```
int vector [ 16 ];
```

```
int result [ 16 ];

MPI_Comm_rank ( MPI_COMM_WORLD, &myRank );
for ( i = 0; i < 16; i++ )
  vector [ i ] = myRank * 100 + I;

MPI_Reduce  (
 vector, // кожний MPI-процес надає свій вектор
 result, // процес з номером "root" збирає результат тут
 16,     // розмір вихідного і результуючого векторів
 MPI_INT,// тип елементів вектора
 MPI_SUM,// колективна операція: поелементне додавання векторів
 0,      // номер процесу "root"
 MPI_COMM_WORLD // комунікатор, у рамках якого здійснюється
                // колективна операція
);
if ( myRank == 0 )
   // друк вектора result
. . .
```

Пояснити поведінку програми (зокрема, вміст вектора result) під час запуску програми з (np – N) за різних N.

Завдання 2. Модифікувати попередню програму, замінивши операцію `MPI_Reduce` на `MPI_Allreduce`, і передбачити друк вмісту вектора result у кожному з процесів. Пояснити отримані результати і запропонувати еквівалентну схему реалізації, що використовує функцію `MPI_Reduce` і функції колективного обміну даними.

Завдання 3. Реалізувати програму, яка:

1) розсилає рядки вихідної прямокутної матриці по процесах;

2) знаходить максимальний елемент в кожному стовпці цієї матриці;

3) перемножує всі елементи *i*-го рядка матриці на максимальний елемент *i*-го стовпця;

4) збирає отриману матрицю в процесі з номером 0, який її роздруковує.

Зміст звіту

1. Назва та мета роботи.

2. Лістинги розроблених програм та результати їх роботи.

3. Письмові відповіді на питання для самоперевірки.

Питання для самоперевірки

1. Особливості реалізації семафорів у бібліотеці Pthread.
2. Чим м’ютекс відрізняється від семафора в бібліотеці Pthread?
3. Як у бібліотеці Pthread використовуються умовні змінні?
4. Основні види блокувань, застосовуваних у бібліотеці Pthread.
5. У чому відмінність і в чому подібність м’ютексів і блокувань?
6. Основні функції для роботи з блокуваннями у бібліотеці Pthread.
7. Навести особливості блокувань читання–запису.

Література: [7, с. 295–311; 9, с. 405–425; 11, с. 679–687; 12, с. 518–526].

Самостійна робота № 5

Тема. Керування процесами в MPI

Мета: вивчити поняття «група» і «комунікатор», засоби їх створення та керування ними у MPI-програмах.

Короткі теоретичні відомості

Типовий сценарій формування нових груп MPI-процесів на основі вихідної глобальної групи MPI_COMM_WORLD та організації взаємодії всередині нових груп складається з таких кроків.

1. Отримати обробник глобальної групи, пов’язаної з комунікатором MPI_COMM_WORLD, використовуючи функцію MPI_Comm_group.
2. Створити нову групу як підмножину глобальної групи, використовуючи функцію MPI_Group_incl.
3. Створити новий комунікатор для новоствореної групи, використовуючи функцію MPI_Comm_create.
4. Отримати новий ранг процесу в новоствореному комунікаторі, використовуючи функцію MPI_Comm_rank.

5. Виконати обмін повідомленнями між процесами в межах новоствореної групи.

6. Після закінчення роботи, звільнити (знищити) групу і комунікатор, використовуючи функції MPI_Group_free і MPPI_Comm_free.

Схематично такий сценарій можна зобразити на рис. 1:

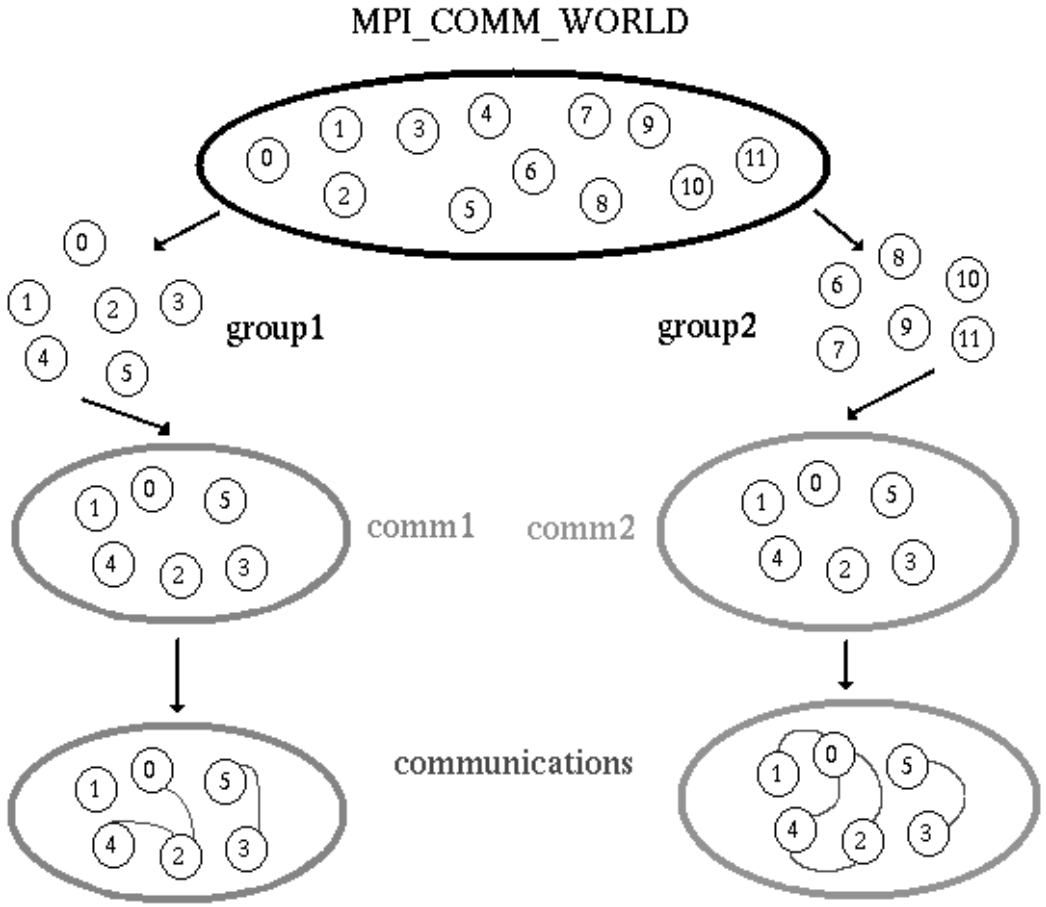

Рисунок 1 – Схема взаємодії MPI-процесів на основі MPI_COMM_WORLD

Порядок виконання самостійної роботи

Завдання 1. Реалізувати програму, у якій організовується дві групи процесів для виконання колективної операції всередині кожної з груп. Запустити на виконання дану програму з ключем -np 8.

Пояснити результати роботи програми, що виводяться на консоль.

```
#include "mpi.h"
#include <stdio.h>
#define NPROCS 8
int main(argc,argv)
```

```
int argc;
char *argv[];  {
int        rank, new_rank, sendbuf, recvbuf, numtasks,
           ranks1[4]={0,1,2,3}, ranks2[4]={4,5,6,7};
MPI_Group  orig_group, new_group;
MPI_Comm   new_comm;
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
if (numtasks != NPROCS) {
  printf("Must specify MP_PROCS= %d. Terminating.\n",NPROCS);
  MPI_Finalize();  exit(0);  }
sendbuf = rank;
/* Extract the original group handle */
MPI_Comm_group(MPI_COMM_WORLD, &orig_group);
/* Divide tasks into two distinct groups based upon rank */
if (rank < NPROCS/2) {
  MPI_Group_incl(orig_group, NPROCS/2, ranks1, &new_group);  }
else {MPI_Group_incl(orig_group, NPROCS/2, ranks2, &new_group);}
/* Create new new communicator and then perform collective
communications */
MPI_Comm_create(MPI_COMM_WORLD, new_group, &new_comm);
MPI_Allreduce(&sendbuf, &recvbuf, 1, MPI_INT, MPI_SUM,new_comm);
MPI_Group_rank (new_group, &new_rank);
printf("rank= %d newrank= %d recvbuf=
%d\n",rank,new_rank,recvbuf);
MPI_Finalize(); }
```

Задача 2. Модифікувати текст попередньої програми таким чином, щоб:

а) у межах першої групи виконувалася колективна операція MPI_MIN, і результат поміщався в процес з найменшим рангом у глобальній (вихідній) групі та роздруковувався ним;

б) у межах другої групи виконувалася колективна операція MPI_MAX, і результат поміщався в процес з найбільшим рангом у глобальній (вихідній) групі та роздруковувався ним.

Зміст звіту

1. Назва та мета роботи.

2. Лістинги розроблених програм і результати їх роботи.

3. Відповіді на питання для самоперевірки.

Питання для самоперевірки

1. Способи поділу об’єктів ядра в Win32 API.
2. Організація процесу в операційній системі MS Windows.
3. Параметри процесу в операційній системі MS Windows.
4. Породження і завершення процесу з використанням Win32 API.
5. Особливості організації та використання потоків у Win32 API.
6. Параметри потоку в операційній системі MS Windows.

Література: [13, с. 415–428; 14, с. 300–317; 17, с. 27–39].

Самостійна робота № 6

Тема. Організація логічних топологій процесів

Мета: вивчити види топологій MPI-процесів – декартові та графові, а також призначення та застосування функцій для їх створення та керування ними.

Короткі теоретичні відомості

У стандарті MPI існує два основних типи віртуальних топологій: декартова (решіткова) і графова.

Обидві ці топології будуються на основі груп і комунікаторів та «програмуються» розробником паралельного додатка.

У програмі, що наведена нижче, 16 процесів об’єднуються в декартову топологію 4 x 4:

0 (0,0)	1 (0,1)	2 (0,2)	3 (0,3)
4 (1,0)	5 (1,1)	6 (1,2)	7 (1,3)
8 (2,0)	9 (2,1)	10 (2,2)	11 (2,3)
12 (3,0)	13 (3,1)	14 (3,2)	15 (3,3)

У програмі кожен процес обмінюється своїм рангом із чотирма сусідніми процесами, де для процесу з координатами (i, j) його сусідами вважаються процеси з такими координатами:

$$(i-1, j),$$
$$(i,\ j-1),$$
$$(i, j+1),$$
$$(i+1, j+1),$$

якщо їх координати не виходять за межі заданої декартової решітки.

Порядок виконання самостійної роботи

Задача 1. Відтранслювати, виконати і пояснити виданий результат для програми, наведеної нижче.

```
#include "mpi.h"
#include <stdio.h>
#define SIZE 16
#define UP    0
#define DOWN  1
#define LEFT  2
#define RIGHT 3
int main(argc,argv)
int argc;
char *argv[];  {
int numtasks, rank, source, dest, outbuf, i, tag=1,
inbuf[4]={MPI_PROC_NULL,MPI_PROC_NULL,MPI_PROC_NULL,MPI_PROC_NULL,
}, nbrs[4], dims[2]={4,4},
   periods[2]={0,0}, reorder=0, coords[2];
MPI_Request reqs[8];
MPI_Status stats[8];
MPI_Comm cartcomm;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
if (numtasks == SIZE) {
  MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, reorder,
&cartcomm);
  MPI_Comm_rank(cartcomm, &rank);
  MPI_Cart_coords(cartcomm, rank, 2, coords);
  MPI_Cart_shift(cartcomm, 0, 1, &nbrs[UP], &nbrs[DOWN]);
  MPI_Cart_shift(cartcomm, 1, 1, &nbrs[LEFT], &nbrs[RIGHT]);
  outbuf = rank;
  for (i=0; i<4; i++) { dest = nbrs[i];  source = nbrs[i];
     MPI_Isend(&outbuf, 1, MPI_INT, dest, tag,
               MPI_COMM_WORLD, &reqs[i]);
     MPI_Irecv(&inbuf[i], 1, MPI_INT, source, tag,
               MPI_COMM_WORLD, &reqs[i+4]); }
```

```
  MPI_Waitall(8, reqs, stats);
printf("rank= %d coords= %d %d  neighbors(u,d,l,r)= %d %d %d
%d\n", rank,coords[0],coords[1],nbrs[UP],nbrs[DOWN],nbrs[LEFT],
nbrs[RIGHT]);
  printf("rank= %d inbuf(u,d,l,r)= %d %d %d %d\n",
        rank,inbuf[UP],inbuf[DOWN],inbuf[LEFT],inbuf[RIGHT]);  }
else  printf("Must specify %d processors. Terminating.\n",SIZE);
MPI_Finalize(); }
```

Задача 2. Нехай для процесу з координатами (i, j) його сусідами вважаються процеси з такими координатами:

$$(i - 1, j - 1),$$
$$(i - 1, j + 1),$$
$$(i + 1, j - 1),$$
$$(i + 1, j + 1).$$

Модифікувати попередню програму так, щоб кожен процес здійснював обмін рангами з новою множиною своїх сусідів.

Зміст звіту

1. Назва та мета роботи.

2. Лістинги розроблених програм і результати їх роботи.

3. Відповіді на питання для самоперевірки.

Питання для самоперевірки:

1. Породження та завершення потоку з використанням Win32 API. Синхронізація потоку в режимі користувача: Interlocked-функції.

2. Синхронізація потоку в режимі користувача: структура CRITICAL_SECTION.

3. Синхронізація потоків за допомогою об’єктів ядра. Використання Wait-функцій.

4. Організація виняткового доступу до ресурсу за допомогою подій.

5. Функції Win32 API для роботи з подією, умовна синхронізація за допомогою подій.

Література: [11, с. 314–340; 12, с. 708–719; 21, с. 56–67].

Самостійна робота № 7

Тема. Розробка складних паралельних програм з використанням MPI

Мета: вивчити паралельний алгоритм Фокса перемноження матриць і реалізувати його із застосуванням MPI; отримати експериментальні дані про масштабованості цього алгоритму.

Короткі теоретичні відомості

У роботі вивчають 2 паралельні алгоритми множення матриць:

– з декомпозицією по рядках;

– з блочною декомпозицією.

Нижче в тексті передбачається, що матриці є квадратними порядку *N*. Тип елементів матриць може бути вибраний довільним – цілочисловий, з точкою, що плаває, з точкою, що плаває, подвійної точності і т. д.

Потрібно реалізувати паралельний алгоритм множення матриць з використанням декомпозиції по рядках. Нехай дані вихідні матриці *A*, *B*, а *C* є результуючою матрицею, тобто, *C* = *A* x *B*.

Передбачається, що для матриць порядку *N* алгоритм виконується на *N* процесорах. Відповідно, *i*-й рядок матриць *A, B* призначається процесору *i*, і він відповідальний за обчислення *i*-го рядка результуючої матриці C.

Суть алгоритму полягає в знаходженні елемента з координатами (*i, j*) результуючої матриці *C* процесом з номером *i* за допомогою обчислення «поточкового» (скалярного) добутку рядка *i* матриці *A* і стовпця *j* матриці *B*. Для обчислення цього добутку, процесу *i* знадобиться весь стовпець *j* матриці *B*, що може бути реалізовано унаслідок застосування операції `MPI_Allgather` у всіх процесах:

```
for  для кожного стовбця b матриці B
{ Allgather ( b );
  Обчислити поточковий добуток рядка i матриці A
  і стовбця b матриці B }
```

Альтернативним застосуванню `MPI_Allgather` може бути розв’язання з попереднім розсилання всієї матриці *B* усім процесам.

Порядок виконання самостійної роботи

Реалізувати паралельний алгоритм множення матриць з блочною декомпозицією (алгоритм Фокса). У цьому алгоритмі передбачається, що матриці розмірності N x N розподіляються між P процесами, які організовані у вигляді декартової решітки (квадрата) зі стороною $\sqrt{P}$.

Передбачається, що N ділиться без остачі на $\sqrt{P}$, а тому кожен процес зберігає підматриці розмірності N x N, де $N = N / \sqrt{P}$.

Наприклад, за $N = 16$, тобто матриць розмірності 16 x 16, і $P = 4$, тобто для решітки процесорів 2 x 2, кожен процес зберігає підматриці розміром 8 x 8 вихідних матриць A, B, і відповідальний за обчислення відповідної підматриці результуючої матриці C.

Нехай A_{ij} є підматрицею розмірності N x N матриці A, першим елементом якої є елемент A [$i * N$, $j * N$] (аналогічно для підматриць B_{ij} і C_{ij}).

Підматриці A_{ij}, B_{ij} і C_{ij} призначаються процесу з номером (i, j). Тоді алгоритм роботи процесу з номером (i, j) такий:

```
q = sqrt ( p );
dest   = ( ( i – 1 ) mod q, j );
source = ( ( i + 1 ) mod q, j );
for ( stage = 0; stage < q; stage++ )
{ k_prime = ( i + stage ) mod q;
  Broadcast A [ i, k_prime ] across process row i;
  C [ j, j ] += A [ i, k_prime ] * B [ k_prime, j ];
  Send B [ k_prime, j ] to dest;
  Receive B [ (k_prime + 1 ) mod q, j ] from source; }
```

Реалізація цього алгоритму у вигляді MPI-програми показана нижче:

```
dest = (my_row + q - 1)% q; // Призначення для циклічного зсуву
                            // у стовпцях
source = ((i + 1) mod q, j); // Джерело для циклічного зсуву
                             // у стовпцях
// Розміщення пам’яті для тимчасової локальної матриці
tmp = (float *) malloc (sizeof (float) * N_local * N_local);
settozero (C_local, N_local); // Занулення результуючої матриці
                              // C_local
for (stage = 0; stage <q; stage ++)
{  bcast_root = (my_row + stage)% q; // Процес, який виконує
                         // Колективну (радіомовну) розсилку
 if (bcast_root == my_col)
 {  // Розіслати всім іншим процесам в цьому самому рядку
  MPI_Bcast  (A_local,  N_local  *  N_local,  MPI_FLOAT,  bcast_root,
row_comm);
```

```
  // Перемножити підматриці
  matrixmult (A_local, B_local, C_local, N_local); }
 else {
  // Прийняти підматрицю матриці A від процесу, який виконує
  // колективну розсилку
  MPI_Bcast (tmp,N_local*N_local,MPI_FLOAT,bcast_root, row_comm);
  // Перемножити її з власною подматріцею B_local
  matrixmult (tmp, B_local, C_local, N_local); }
 // Послати підматрицю матриці B наверх і прийняти нову підматрицю
 // знизу
 MPI_Sendrecv_replace (B_local, N_local * N_local, MPI_FLOAT,
dest, datatag, source, datatag, col_comm, & status); }
```

Відтранслювати і виконати цю програму для матриць розмірності 100x100 з використанням 16 процесорів (-np 16).

Зміст звіту

1. Назва та мета роботи.

2. Лістинги розроблених програм і результати їх роботи.

3. Відповіді на питання для самоперевірки.

Питання для самоперевірки

1. Поняття семафора. Організація виняткового доступу до ресурсу за допомогою подій.

2. Функції Win32 API для роботи із семафорами. Умовна синхронізація за допомогою семафорів.

3. Поняття м’ютекса. Організація виняткового доступу до ресурсу за допомогою м’ютексів.

4. Функції Win32 API для роботи з м’ютексами.

Література: [3, с. 480–499; 6, с. 160–175; 7, с. 280–292, 8, с. 98–118].

Самостійна робота № 8

Тема. Налагодження MPI-програм з використанням Visual Studio 2005

Мета: вивчити настройку конфігураційних параметрів для налагодження MPI-додатків у рамках Visual Studio 2005; вивчити способи задання точок зупинки (breakpoints) для окремих і множин MPI-процесів.

Короткі теоретичні відомості

Налагодження MPI-програм з використанням Visual Studio +2005 складається, у загальному випадку, з трьох кроків:

1) запуск віддаленого наладника (remote debugger) на вузлах, зарезервованих для сеансу налагодження;

2) конфігурування налагоджувальних властивостей проекту MPI-додатки в рамках Visual Studio;

3) запуск програми в режимі налагодження та перегляду процесів у точках їх зупинки (breakpoints).

Порядок виконання самостійної роботи

Задача 1. Запустити віддалений наладник msvsmon.exe на декількох вузлах (припустимо, що мають імена Node1, Node2, Node3, Node4), сформувавши і запустивши наступне завдання з використанням інтерфейсу командного рядка:

```
job new /jobname:ReserveTimeForDebug
/askednodes:Node1,Node2,Node3,Node4 /runtime:00:05:00
/rununtilcancelled:true /scheduler:myCluster
job add <JobID> /requirednodes:Node1 C:\Program Files\Microsoft
Visual Studio 8\Common7\IDE\Remote Debugger\x64\msvsmon.exe
job add <JobID> /requirednodes:Node2 C:\Program Files\Microsoft
Visual Studio 8\Common7\IDE\Remote Debugger\x64\msvsmon.exe
job add <JobID> /requirednodes:Node3 C:\Program Files\Microsoft
Visual Studio 8\Common7\IDE\Remote Debugger\x64\msvsmon.exe
job add <JobID> /requirednodes:Node4 C:\Program Files\Microsoft
Visual Studio 8\Common7\IDE\Remote Debugger\x64\msvsmon.exe
job submit /id:<JobID>
```

Задача 2. Як приклад налагоджують програму, тут передбачається програма перемноження матриць з декомпозицією по рядках.

Необхідно відкрити дану програму в рамках Visual Studio, і на сторінці Properties проекту вибрати послідовно Configuration Properties -> Debugging, і в списку Debugger to launch вибрати MPI Cluster Debugger.

На цій самій сторінці необхідно заповнити відповідні поля (MPIRun Command, MPIRun Arguments, MPIShim location та ін.) потрібними значеннями для старту програми.

З основного меню Visual Studio +2005 вибрати послідовно Tools -> Options, щоб відкрити діалогове вікно для завдання точок зупинки (breakpoints).

Необхідно встановити точку зупинки в програмі після виконання команди MPI_Allgather і заново побудувати додаток.

Задача 3. Запустити побудований додаток у режимі налагодження, натиснувши F5. Після зупинки програми натиснути Ctrl + Alt + Z для відкриття вікна Processes. Вибрати один з паралельних процесів і проглянути значення його змінних.

Перевірити покрокове виконання програми, починаючи з точки зупинки, використовуючи відповідні клавіші на панелі вікна Processes (рис. 2):

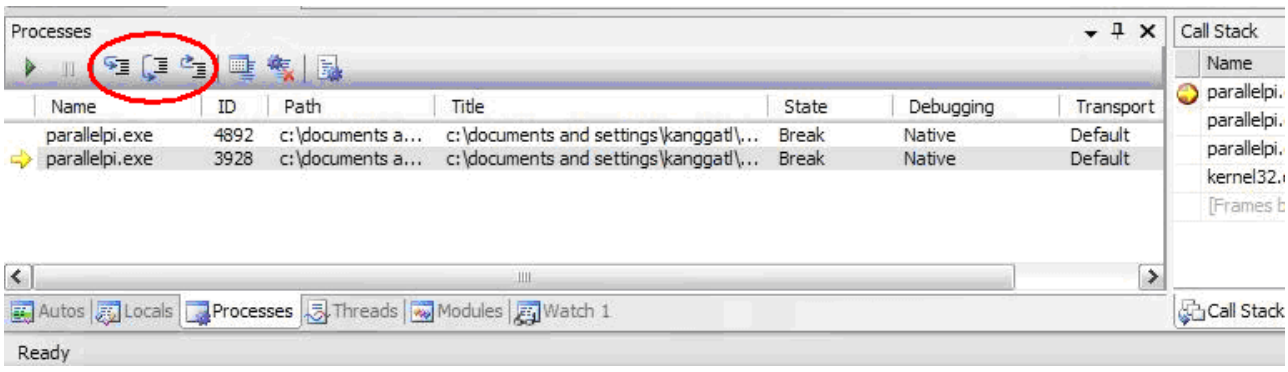

Рисунок 2 – Панель вікна Processes для покрокового виконання програми

Зміст звіту

1. Назва та мета роботи.
2. Лістинги розроблених програм і результати їх роботи.
3. Відповіді на питання для самоперевірки.

Питання для самоперевірки

1. Три основних ознаки MPI-функцій: блокувальні/неблокувальні, локальні, колективні функції.

2. Надати визначення групи, галузі зв’язку, комунікатора.

3. MPI-функції створення комунікаторів.

4. Обрамовувальні та інформаційні функції. Структура найпростішої MPI-програми.

5. Парні функції приймання/передавання повідомлень.

6. Блокувальні операції обміну.

7. MPI-функції для отримання інформації про повідомлення.

8. Неблокувальні операції обміну.

9. Перевірка виконання обміну.

10. Передавання повідомлення в автоматичному режимі (блокувальна/ неблокувальна).

11. Передавання повідомлення з буферизацією (блокувальна/ неблокувальна).

12. Синхронне передавання повідомлення (блокувальна/неблокувальна).

14. Передавання повідомлення за готовністю (блокувальна/ неблокувальна).

15. Реалізація відкладених обмінів.

16. Поєднання операцій посилання та передавання.

17. Функції колективної взаємодії процесів: синхронізація бар’єром.

18. Функції колективної взаємодії процесів: глобальні функції зв’язку

19. Функції колективної взаємодії процесів: глобальні функції приведення

20. Попереднє визначення константи і типи даних у MPI.

Література: [7, с. 330–347; 11, с. 651–670, 13, с. 590–611].

3 КРИТЕРІЇ ОЦІНЮВАННЯ ЯКОСТІ ВИКОНАННЯ САМОСТІЙНИХ РОБІТ СТУДЕНТАМИ

У 8-му студенти виконують 8 самостійних робіт. Загальна кількість балів, яку отримують студенти за виконання самостійних робіт за обидва семестри, становить 24 бали – сума за захист виконаних самостійних робіт (максимально по 3 бали на кожну самостійну роботу).

Шкала оцінювання: національна та ECTS

<table>
<tr><td rowspan="2">Сума балів за всі види навчальної діяльності</td><td rowspan="2">Оцінка ECTS</td><td colspan="2">Оцінка за національною шкалою</td></tr>
<tr><td>Для екзамену, курсового проекту (роботи), практики</td><td>Для заліку</td></tr>
<tr><td>90–100</td><td>A</td><td>Відмінно</td><td rowspan="5">Зараховано</td></tr>
<tr><td>82–89</td><td>B</td><td rowspan="2">Добре</td></tr>
<tr><td>74–81</td><td>C</td></tr>
<tr><td>64–73</td><td>D</td><td rowspan="2">Задовільно</td></tr>
<tr><td>60–63</td><td>E</td></tr>
<tr><td>35–59</td><td>FX</td><td>Незадовільно з можливістю повторного складання</td><td>не зараховано з Можливістю повторного складання</td></tr>
<tr><td>0–34</td><td>F</td><td>Незадовільно з обов’язковим повторним вивченням навчальної дисципліни</td><td>Не зараховано з обов’язковим повторним вивченням навчальної дисципліни</td></tr>
</table>

СПИСОК ЛІТЕРАТУРИ

1. Бекон Д. Операционные системы / Д. Бекон, Е. Харрис. – Киев : Издательская группа BHV, 2004. – 800 с.
2. Вальковский В. А. Распараллеливание алгоритмов и программ. Структурный подход / В. А. Вальковский. – М. : Радио и связь, 1989. – 176 с.
3. Воеводин В. В. Параллельные вычисления / В. В. Воеводин. – СПб. : БХВ-Петербург, 2002. – 608 с.
4. Гергель В. П. Основы параллельных вычислений для многопроцессорных вычислительных систем : учебное пособие / В. П. Гергель, Р. Г. Стронгин. – Нижний Новгород : Изд-во ННГУ, 2003. – 184 с.
5. Кейслер С. Проектирование операционных систем для малых ЭВМ / С. Кейслер. – М. : Мир, 1986. – 680 с.
6. Корнеев В. Д. Параллельное программирование в MPI / В. Д. Корнеев. – Новосибирск : Изд-во СО РАН, 2000. – 213 с.
7. Немнюгин С. А. Параллельное программирование для многопроцессорных вычислительных систем / С. А. Немнюгин, О. Л. Стесик. – СПб. : БХВ-Петербург, 2002. – 400 с.
8. Ортега Дж. Введение в параллельнные и векторные методы решения линейных систем / Дж. Ортега. – М. : Мир, 1991. – 367 с.
9. Рихтер Дж. Windows для профессионалов: создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows / Дж. Рихтер. – СПб. : «Русская редакция», 2000. – 752 с.
10. Стивенс У. Unix: Разработка сетевых приложений / У. Стивенс. – СПб. : Питер, 2004. – 1086 с.
11. Таненбаум Э. Распределенные системы. Принципы и парадигмы. Серия «Классика Computer Science» / Э. Таненбаум, В. Стеен. – СПб. : Питер, 2003. – 877 с.

12. Таненбаум Э. Современные операционные системы / Э. Таненбаум. – СПб. : Питер, 2007. – 1040 с.
13. Хьюз К. Параллельное и распределенное программирование на C++ / К. Хьюз, Т. Хьюз. – М. : Издательский дом «Вильямс», 2004. – 672 с.
14. Эндрюс Г. Р. Основы многопоточного, параллельного и распределенного программирования / Г. Р. Эндрюс. – М. : Издательский дом «Вильямс», 2003. – 512 с.
15. Вахалия Ю. UNIX изнутри / Ю. Вахалия. – СПб. : Питер, 2003. – 844 с.
16. Гольдштейн А. Б. Softswitch. / А. Б. Гольдштейн, Б. С. Гольдштейн. – СПб. : БХВ-Петербург, 2006. – 368 с.
17. Лупин С. А. Технологии паралельного программирования / С. А. Лупин, М. А. Посыпкин. – СПб. : Питер, 2011. – 208 с.
18. Хаггарти Р. Дискретная математика для программистов / Р. Хаггарти. – СПб. : БХВ-Петербург, 2012. – 395 с.
19. Голицына О. А. Программирование на языках высокого уровня / О. А. Голицына, И. И. Попов. – СПб. : Питер, 2010. – 418 с.
20. Вальковский В. А. Синтез паралельних программ и систем на вычислительных моделях / В. А. Вальковский, В. Э. Малышкин. – Новосибирск : Изд-во «Наука», 1988. – 129 с.
21. Малышкин В. Э. Паралельное программирование мультикомпьютеров / В. Э. Малышкин, В. Д. Корнеев. – Новосибирск : Изд-во НТУ, 2006. – 296 с.

Методичні вказівки щодо виконання самостійних робіт з навчальної дисципліни «Паралельні та розподілені обчислення» для студентів денної форми навчання зі спеціальності 123 – «Комп'ютерна інженерія»

Укладач к. т. н., доц. О. Г. Славко

Відповідальний за випуск зав. кафедри КІС А. В. Луговой

Підп. до др. _____. Формат 60×84 1/16. Папір тип. Друк ризографія.
Ум. друк. арк. ____. Наклад _____ прим. Зам. №_____. Безкоштовно.

Видавничий відділ
Кременчуцького національного університету
імені Михайла Остроградського
вул. Першотравнева, 20, м. Кременчук, 39600