Національний технічний університет України  
«Київський політехнічний інститут»

Кафедра обчислювальної техніки

(повна назва кафедри, циклової комісії)

Члени комісії

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Студента 3 курсу групи ІП-31

напряму підготовки 6.050103 «Програмна інженерія»

Кобилинський Д.А.

(прізвище та ініціали)

Керівник доцент Корочкін О.В.

Національна оцінка

Кількість балів:

Оцінка: ECTS

**Курсова робота**

з дисципліни «Паралельні та розподілені обчислення»

(назва дисципліни)

на тему: «Розробка програмного забезпечення   
для паралельних комп’ютерних систем»

Київ – 2016 рік

Національний технічний університет України  
«Київський політехнічний інститут»

Факультет (інститут) інформатики та обчислювальної техніки

(повна назва)

Кафедра обчислювальної техніки

(повна назва)

Освітньо-кваліфікаційний рівень бакалавр

Напрям підготовки 6.050103 «Програмна інженерія»

(шифр і назва)

**Завдання**

на курсову роботу студенту

Кобилинський Дмитро Анатолійович

(прізвище, ім’я, по батькові)

1. Тема роботи «Розробка програмного забезпечення для паралельних комп’ютерних систем».
2. Керівник роботи к. т. н., доцент Корочкін Олександр Володимирович.

(вчене звання, науковий ступінь, прізвище та ініціали)

1. Строк подання студентом роботи 18 травня 2016 р.
2. Вихідні дані для роботи:

* огляд засобів роботи з процесами в бібліотеці PVM;
* математична задача МА = МВ\*МС
* структури ПКС СП та ПКС ЛП;
* бібліотеки програмування: Java, Ada Рандеву.

1. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

* огляд засобів роботи з процесами в бібліотеці PVM;
* розробка і тестування програми ПРГ1 для ПКС СП;
* розробка і тестування програми ПРГ2 для ПКС ЛП.

1. Перелік графічного матеріалу:

* структурна схема ПКС СП;
* структурна схема ПКС ЛП;
* схеми алгоритмів процесів і головної програми для ПРГ1;
* схеми алгоритмів процесів і головної програми для ПРГ2.

1. Дата видачі завдання

Календарний план

|  |  |  |
| --- | --- | --- |
| №  з/п | Назва етапів виконання КР | Строк виконання етапів КР |
| 1 | Виконання Розділу 1 | 01.03.2016 |
| 2 | Виконання Розділу 2 | 23.03.2016 |
| 3 | Виконання Розділу 3 | 23.04.2016 |
| 4 | Тестування програм ПРГ1 та ПРГ2 | 10.05.2016 |
| 5 | Оформлення КР | 17.05.2016 |
| 6 | Захист КР | 18.05.2016 |

Студент

(підпис) (прізвище та ініціали)

Керівник роботи

(підпис) (прізвище та ініціали)

ЗМІСТ

[РОЗДІЛ 1. ОГЛЯД ЗАСОБІВ РОБОТИ З ПРОЦЕСАМИ В БІБЛІОТЕЦІ PVM 3](#_Toc444441356)

[1.1 Загальна інформація 3](#_Toc444441357)

[1.2 Загальна структура простої програми 3](#_Toc444441358)

[1.3 Створення нової задачі 3](#_Toc444441359)

[1.4 Передача повідомлень 3](#_Toc444441360)

[1.5 Робота з буферами 3](#_Toc444441361)

[1.6 Пакування і розпакування повідомлень 3](#_Toc444441362)

[1.7 Передавання і приймання повідомлень 3](#_Toc444441363)

[1.8 Робота з групами 3](#_Toc444441364)

[1.9 Приклад використання 3](#_Toc444441365)

[1.10 Висновки по розділу 1 3](#_Toc444441368)

[СПИСОК ….ДЖЕРЕЛ 3](#_Toc444441369)

**ВСТУП**

Паралельні обчислювальні системи - це фізичні комп'ютерні, а так само програмні системи, що реалізують тим чи іншим способом паралельну обробку даних на багатьох обчислювальних вузлах.

Ідея розпаралелювання обчислень базується на тому, що більшість завдань може бути розділене на набір менших завдань, які можуть бути вирішені одночасно. Головна мета використання паралельних обчислень - це підвищення швидкості обчислень за рахунок їх паралельного виконання.

У даній курсовій роботі розглядаються засоби роботи з процесами в одній з найпопулярніших систем паралельного програмування: PVM (Parallel Virtual Machine - паралельна віртуальна машина).

# 

# РОЗДІЛ 1. ОГЛЯД ЗАСОБІВ РОБОТИ З ПРОЦЕСАМИ В БІБЛІОТЕЦІ PVM

## Загальна інформація

PVM (Parallel Virtual Machine) є побічним продуктом науково-дослідницького проекту, пов’язаного з гетерогенними розрахунками [1]. Головними цілями цього проекту є дослідження гетерогенних розрахунків і розробка зручних, універсальних засобів для організацію гетерогенних розрахунків з використанням комп’ютерів різної архітектури. PVM представляє собою інтегрований набір бібліотек і програмних інструментів для організації гетерогенних розрахунків з використанням зв’язаних між собою обчислювальних пристроїв різноманітних архітектур. Загальна мета PVM – зв’язування цих різноманітних систем для одночасних і паралельних розрахунків.

* Основні принципи PVM:
* Користувацький пул хостів – розрахунок задачі виконується на множині комп’ютерів, яка задається користувачем для запуску даної PVM програми. Як і однопроцесорні системи, так і багатопроцесорні системи(з різними типами пам’яті) можуть бути частиною пулу. Пул може бути змінений під час виконання програми, з нього можуть бути видалені, або додані нові хости(це забезпечує надійність розрахунків під час відмови одного з хостів)
* Прозорий доступ до обладнання – прикладні програми можуть розглядати обладнання як колекцію віртуальних процесорних елементів, або можуть обирати хости з конкретними, найбільш підходящими, апаратними засобами.
* Обчислення на основі процесів – одиниця паралелізму в PVM є задача(найчастіше Unix процес), незалежний послідовний потік управління, який працює в режимі розрахунку або комунікації. Ніякої відповідності задача – процесор PVM встановлює(можливі випадки, коли декілька задач виконуються на одному і тому ж процесорі)
* Модель явної передачі повідомлень – набір задач, кожна з яких виконує свою частину обчислень, явно взаємодіють через відсилання і приймання повідомлень один одному. Розмір повідомлень обмежений лише об’ємом доступної пам’яті.
* Підтримка неоднорідності – система PVM підтримує неоднорідність в плані машин, мереж, програм. Що до передачі повідомлень, PVM дозволяє передавати декілька типів даних між машинами, які мають різне представлення даних.
* Підтримка багатопроцесорних систем – система PVM може використовувати засоби передачі хоста, на якому виконується програма. Таким чином можливе використання переваг обладнання і операційної системи. Деякі розробники представляють власні, оптимізовані для їхніх систем, реалізації PVM, які можуть бути зв’язані з публічної версією PVM.

Система PVM складається з двох частин. Перша частина являє собою демон, який носить назву *pvmd3*, інколи скорчено *pvmd* . Демони знаходяться на всіх комп’ютерах, які входять в склад віртуальної машини PVM. Перед запуском PVM програми користувач має запустити демон і створити віртуальну машину PVM. Користувач може створити декілька віртуальних машин і виконувати декілька програм одночасно.

НЕ СОБЛЮДАЕТСЯ 1,5 интервал!!!!

Інша частина являє собою бібліотеку інтерфейсу PVM. Вона включає в себе набір примітивів, які необхідні для комунікації між задачами і програмами. Ця бібліотека містить набір функції, які викликаються користувачем, для передачі повідомлень, створення задач і їх координації, а також функції для зміни структури віртуальної машини PVM.

Модель розрахунків PVM заснована на тому, що програма складається з декількох задач. Кожна задача відповідає за частину розрахункового навантаження програми. Інколи програму розділяють за функціями, тобто кожна задача виконую свою окрему функцію(введення даних, розрахунок, вивід даних…). Цей метод носить назву функціонального паралелізму. Але більш поширеним способом організації обчислень є паралелізм даних. В цьому методі всі задачі дуже подібні, але кожна з них оброблює лише не велику частину даних.

Система PVM підтримує мови програмування С, С++ та Фортран. Вибір мов програмування був зроблений на основі досліджень, які показали, що дані мови найчастіше використовуються для складних розрахунків.

Для С та С++ прив’язки інтерфейсу PVM реалізовані у вигляді функцій у відповідності з прийнятими стандартами. А якщо точніше, аргументи функцій являють собою значення параметрів і вказівники(при необхідності) а результатом функції є статус виконання виклику функції. Також є набір макросів і функції для визначення типу результату. Для використання PVM в мовах С та С++ використовується бібліотека *libpvm3.a*, що є частиною стандартної поставки.

В мові Фортран прив’язка виконана у вигляді підпрограм, а не функцій. Це пов’язано з тим. що деякі компілятори не можуть надійно підтримувати роботу функції. Тому вводиться додатковий аргумент підпрограм, через який в основну програму повертається результат виклику функцій PVM. Для використання в мові Фортран необхідно мати бібліотеку *libfpvm3.a* і бібліотеку для мови С(*libpvm3.a*).

Всі задачі в PVM, незалежно від обраної мови програмування, мають унікальний цілочисельний ідентифікатор задачі(TID). Повідомлення передаються і отримуються за допомогою TID. TID є унікальним у всій віртуальній машині PVM, і задаються *pvmd*, а не користувачем.

Існують програми, де необхідно використати *групу задач*. І є випадки, коли користувач хоче визначати свої задачі по номерам 0-(p-1), де p – кількість задач. PVM включає концепцію груп, іменованих користувачем. Коли задача приєднується до такої групи, їй присвоюється унікальний номер в межах цієї групи. Згідно з філософією PVM, групи є засобом для об’єднання дуже спільних задач. Різні групи можуть перекриватись, і задачі, які входять в декілька груп, можуть використовувати для передачі повідомлень з однієї групи в іншу.

В загальному розробка програми на PVM виглядає так: користувач створює програму, використовуючи будь яку доступну мову, записує об’єктні файли на хости, які входять до пулу. Ці об’єктні файли є унікальними для різних архітектур. Далі, як правило, користувач запускає одну задачу(головну, або початкову), яка в свою чергу запускає наступні необхідні задачі.

## 1.2 Загальна структура простої програми с отступом!!!!

Перш за все необхідно включити в програму опис функцій PVM. В мові С це робиться за допомогою директиви препроцесора *include*(*#include "pvm3.h"* ). Також необхідно зберігати TID поточної задачі і MSGTAG(аналог TID для повідомлень). Обидва параметри мають тип int. Якщо використовується головна задача, то вона має містити TID всіх задач, які будуть створені нею.

Для обміну текстовими повідомленнями необхідно використовувати буфер, пам'ять під якого має бути виділена перед початком виконання задачі. В мові С зазвичай буфером є масив з елементів типу char. Розмір масиву обирається в залежності від максимального розміру повідомлення. Інколи, для забезпеченні можливості легкого розширення програми, розмір буферу визначають з запасом.

Для керування правильністю виконання програми необхідно використати ще одну цілочисельну змінну, в яку буде записуватись результат виконання функцій. Викликаючи будь яку функцію, ми зберігаємо результат її виклику. Далі в програмі ми перевіряємо цей результат, і згідно з ним корегуємо виконання програми.

При використанні механізму іменованих груп необхідно зберігати ідентифікатори груп, а також, в залежності від програми, відповідність задач до груп.

## 1.3 Створення нової задачі

Функція для створення нової задачі має сигнатуру:

int numt = pvm\_spawn( char \*task, char \*\*argv, int flag, char \*where, int ntask, int \*tids )

Параметри:

* task – шлях до виконуваного файлу програми. Шлях може бути абсолютним, або відносним до шляху пошуку PVM.
* argv – масив аргументів, яку будуть передані до створеної задачі(якщо підтримується хостом). Якщо необхідності в аргументах немає, параметр має бути NULL.
* flag – ціле число, яке вказує на параметри створення, є сумою наступних параметрів
  + PvmTaskDefault – PVM обирає будь який хост для створення задачі
  + PvmTaskHost – вказати певний хост
  + PvmTaskArch – вказати певну архітектуру
  + PvmMppFront – запустити задачу на хості МПП архітектурі
* where – рядок, який вказує, де запустити задачу. В залежності від flag це може бути назва хоста(ibm1.epm.ornl.gov'), назва архітектури(SUN4). Якщо flag рівний 0, то where ігнорується, і задача буде запущена на будь якому вільному хості
* ntask – кількість задач для запуску
* tids – вказівник на масив, в який будуть записані ідентифікатори створених задач.
* numt – статус виконання створення задач. Якщо значення менше 0, то виникла помилка при створенні, якщо більше 0 але менше ntask, то при створенні деяких задач виникла помилка, в такому випадку користувач повинен перевіряти tids на коди помилок.

Коди помилок:

* PvmBadParam – не правильне значення параметру функції
* PvmNoHost – вказаний хост не є віртуальною машиною
* PvmNoFile – не знайдено виконавчий файл програми
* PvmNoMem – недостатньо пам’яті на хості
* PvmSysErr – pvmd не відповідає на запити
* PvmOutOfRes – недостатньо ресурсів

Приклад використання функції:

numt = pvm\_spawn( "host", 0, PvmTaskHost, "sparky", 1, &tid[0] );

Тут створюється задача з файлу host, без використання аргументів, на архітектурі sparky в кількості 1 штука. Ідентифікатор задачі буде збережено в нульовому елементі масиву tid.

## Передача повідомлень

Засобом взаємодії задач в PVM є обмін повідомленнями. Існує ціла низка функцій для роботи з повідомленнями, і буферами, що використовуються для зберігання повідомлень. ПОДРАЗДЕЛ ИЗ ОДНОГО ПРЕДЛОЖЕНИЯ ??!!

## Робота з буферами

Для ініціалізації відправки повідомлень використовується функція pvm\_initsend.

Сигнатура:

int bufid = pvm\_initsend( int encoding )

Параметри:

* encoding – кодування повідомлень
  + PvmDataDefault – автоматичне кодування даних
  + PvmDataRaw – кодування відсутнє
  + PvmDataInPlace – дані будуть упаковані і передані в буфер задачі, що приймає, минаючи буфер задачі, що передає
* bufid – ідентифікатор створеного буфера

Коди помилок:

* PvmBadParam – неправильний параметр кодування
* PvmNoMem – недостатньо пам’яті для буфера

Для створення нового порожнього буфера використовується функція pvm\_mkbuf.

Сигнатура:

int bufid = pvm\_mkbuf( int encoding )

Параметри:

* encoding – кодування повідомлень
  + PvmDataDefault – автоматичне кодування даних
  + PvmDataRaw – кодування відсутнє
  + PvmDataInPlace – дані будуть упаковані і передані в буфер задачі, що приймає, минаючи буфер задачі, що передає
* bufid – ідентифікатор створеного буфера

Коди помилок:

* PvmBadParam – неправильний параметр кодування
* PvmNoMem – недостатньо пам’яті для буфера

Для знищення буферу використовується функція pvm\_freebuf.

Сигнатура:

int info = pvm\_freebuf( int bufid )

Параметри:

* bufid – ідентифікатор буфера для знищення
* info – код виконання функції, якщо менше 0, виникла помилка

Для отримання активного буферу передавання(приймання) використовується функція pvm\_getsbuf (pvm\_getrbuf).

Сигнатура:

int bufid = pvm\_gets(r)buf ()

Параметри:

* bufid – ідентифікатор активного буфера передавання(приймання)

Для переведення активності з одного буферу передавання(приймання) на інший використовується функція pvm\_setsbuf (pvm\_setrbuf).

Сигнатура:

int oldBuf = pvm\_sets(r)buf (int bufid)

Параметри:

* bufid – ідентифікатор нового буфера передавання(приймання)
* oldBuf – ідентифікатор старого буфера передавання(приймання)

## Пакування і розпакування повідомлень

Всі функції пакування і розпакування повідомлень однотипні, і відрізняються лише типом даних, які вони пакують.

Функція для пакування типу float.

Сигнатура:

int info = pvm\_pkcplx( float \*cp, int nitem, int stride )

Параметри:

* cp – вказівник на масив
* nitem – кількість елементів масиву
* stride – кількість елементів для пропуску при пакуванні

## Передавання і приймання повідомлень

Функція передавання повідомлення - сдвиг!!!!

Сигнатура:

int info = pvm\_send( int tid, int msgid)

Параметри:

* tid – TID задачі, до якої буде відправлене повідомлення
* msgid – ідентифікатор повідомлення
* info – статус виконання

Функція масового передавання повідомлення

Сигнатура:

int info = pvm\_mcast( int \*tids, int ntask, int msgid)

Параметри:

* tids – масив TID задач, до яких буде відправлене повідомлення
* ntask – кількість задач в масиві
* msgid – ідентифікатор повідомлення
* info – статус виконання

Функція блокованого приймання повідомлення(задача буде заблокована до моменту прийняття повідомлення). Повідомлення записується в активний буфер.

Сигнатура:

int bufid = pvm\_recv( int tid, int msgtag )

Параметри:

* tid –TID задачі, з якої необхідно прийняти повідомлення
* msgtag – ідентифікатор повідомлення
* info – статус виконання

Функція не блокованого приймання повідомлення

Сигнатура:

int bufid = pvm\_nrecv( int tid, int msgtag )

Параметри:

* tid –TID задачі, з якої необхідно прийняти повідомлення
* msgtag – ідентифікатор повідомлення
* bufid – ідентифікатор буфера, в який буде записано прийняте повідомлення

Функція часового приймання повідомлення

Сигнатура:

int bufid = pvm\_trecv( int tid, int msgtag, struct timeval \*tmout )

Параметри:

* tid –TID задачі, з якої необхідно прийняти повідомлення
* msgtag – ідентифікатор повідомлення
* tmout – часовий інтервал, впродовж якого буде очікуватись повідомлення
* bufid – ідентифікатор буфера, в який буде записано прийняте повідомлення

Функція перевірки на наявність повідомлення

Сигнатура:

int bufid = pvm\_probe( int tid, int msgtag )

Параметри:

* tid –TID задачі, з якої необхідно прийняти повідомлення
* msgtag – ідентифікатор повідомлення
* bufid – ідентифікатор буфера, в який буде записано прийняте повідомлення, при умові що воно вже надійшло

## Робота з групами

Функція для додавання задачі до групи

Сигнатура:

int inum = pvm\_joingroup( char \*group )

Параметри:

* group – назва групи
* inum – номер задачі в групі

Функція для покидання групи

Сигнатура:

int info = pvm\_lvgroup( char \*group )

Параметри:

* group – назва групи
* info – статус виконання функції

## Приклад використання

## pvm\_initsend(PvmDataDefault);

## pvm\_pkint(&num\_data, 1, 1);

pvm\_pkint(&a[num\_data\*i], num\_data, 1);

pvm\_send(task\_ids[i], 4);

В даному прикладі ми ініціалізували стандартне кодування повідомлень, запакували одне ціле число, масив цілих чисел, і відправили повідомлення до задачі task\_ids[i] з ідентифікатором повідомлення 4.

## Висновки по розділу 1

Виконано загальний огляд бібліотеки PVM. Визначено, що PVM став результатом дослідницьких робіт з гетерогенних обчислень. Показано, що PVM має засоби для організації обчислень методом відправки повідомлень; має гнучку структуру, що дозволяє будувати неоднорідні системи; має засоби для організації розрахунків з використанням машин різноманітної архітектури.

Виконано огляд моделі задач PVM. Визначено, що задачі є одиницею паралелізму в PVM. Показано, що задачі мають гнучку систему обміну повідомленнями і об’єднання в групи.

На основі огляду системи обміну повідомленнями, можна зробити висновок, що PVM має всі необхідні засоби для створення паралельних програм будь якої складності; PVM має необхідні засоби для вирішення будь яких задач синхронізації.

# СПИСОК ЛІТЕРАТУРИ

1. Geist A. PVM. Parallel Virtual Machine. A Users' Guide and Tutorial for Network Parallel Computing / Geist A., Dongarra J., Beguelin A.,1965 – 299

3

4

5.

6.

И ВСЕ?