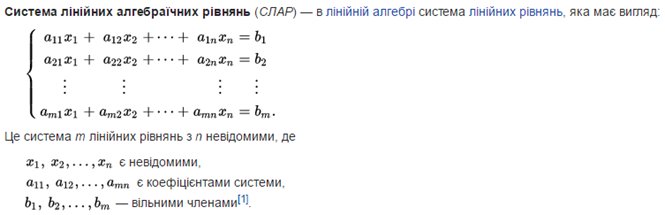
Питання на екзамен з предмету

«Паралельні та розподілені обчислення»

### **1. Що являє собою система лінійних рівнянь? Які типи систем вам відомі? Які методи можуть бути використані для вирішення систем різних типів?**



|  |  |
| --- | --- |
| |  | | --- | | Однорідна(якщо всі вільні члени =0), неоднорідна(якщо хоч один з вільних членів відмінний від 0), сумісна(якщо має хоч один розв’язок), несумісна(якщо жодного розв’язку), визначена(сумісна, що має єдиний розв’язок), невизначена(сумісна, що має безліч розв’язків) | |

**Точні методи**

До точних методів належать методи, що дають точний результат у припущенні ідеальної арифметики. Точні методи можна застосовувати й тоді, коли коефіцієнти й вільні члени рівняння задані в аналітичній, символьній формі.

· **Метод послідовного виключення**. Найпростішим, хоча важким для практичних застосувань, методом розв'язування системи лінійних алгебраїчних рівнянь є метод послідовного виключення невідомих. Суть його в тому, що із першого рівняння змінна x виражається через інші змінні, й підставляється в усі інші рівняння. Це можна зробити, якщо коефіцієнт a відмінний від нуля. У випадку, якщо він нульовий, можна вибрати інше рівняння, оскільки перестановка рівнянь у системі дає еквівалентну систему. В результаті утворюється нова система рівнянь, в якій рівнянь на одне менше. З цією системою рівнянь можна поступити так само, отримуючи ще меншу систему рівнянь. Продовжуючи так, отримують одне лінійне рівняння, з якого можна визначити одну із змінних, а інші, виключені, виразити через неї.

· Метод Гауса — метод, найчастіше застосовуваний при ручному розв'язку СЛАР.

· Метод Гауса-Жордана - модифікація методу Гауса.

· Метод Крамера (за формулами Крамера) — чисто теоретичний метод, непридатний до практичного використання через обчислювальну складність і малу точність, оскільки вимагає обчислення визначників, а тільки в одному визначнику n! доданків. Метод Крамера може застосовуватися для матриць 2×2, або, щонайбільше, 3×3.

· Матричний метод (за допомогою оберненої матриці) - певна теоретична абстракція всіх інших точних методів.

· Метод квадратного кореня — квадратичний метод, який вимагає симетричної матриці системи.

· Метод прогонки зручний для розв'язку систем з тридіагональною матрицею, які часто виникають в задачах математичної фізики.

**Ітераційні методи**

Ітераційні методи встановлюють процедуру уточнення певного початкового наближення до розв’язку. При виконанні умов збіжності вони дозволяють досягти будь-якої точності просто повторенням ітерацій. Перевага цих методів у тому, що вони часто дозволяють досягти розв’язку з наперед заданою точністю швидше, а також розв’язувати більші системи рівнянь. Суть цих методів полягає в тому, щоб знайти нерухому точку матричного рівняння:

x = A’x + b’

еквівалентного початковій системі лінійних алгебраїчних рівнянь.

Серед ітераційних методів можна відзначити найпопулярніші:

· Метод Якобі (метод простої ітерації);

· Метод Зейделя (інколи називають метод Гауса-Зейделя);

· Метод релаксації;

· Багатосітковий метод;

· Метод Монтанте;

· Метод Абрамова (використовується для розв'язування невеликих систем);

· Метод узагальнення мінімальних лишків;

· Метод біспряжених градієнтів;

· Стабілізований метод біспряжених градієнтів;

· Квадратичний метод спряжених градієнтів;

· Метод квазі-мінімальних лишків.

### **2. У чому полягає постановка завдання рішення системи лінійних рівнянь?**

У знаходженні всіх розв’язків системи, або доведенні що система несумісна (Тобто у неї немає розв’язків, які б задовольняли її умову)

\\уявлення не маю, що тут інше можна написати, якщо щось знаєте, то виправте.

### **3. У чому ідея паралельної реалізації методу Гауса?**

Метод Гауса є узагальненням способу підстановки і складається з послідовного виключення невідомих до тих пір, поки не залишиться одне рівняння з одним невідомим. При цьому матриця СЛАР приводиться до трикутного вигляду, де нижче головної діагоналі розташовуються тільки нулі.  
Приведення матриці до трикутного вигляду називається прямим ходом методу Гауса. Зворотний хід починається з рішення останнього рівняння і закінчується визначенням першого невідомого.

Оскільки розв’язання методом Гауса зводиться до послідовності однотипних обчислювальних операцій множення і додавання над рядками матриці, в якості підзадачі можна прийняти обчислення, пов'язані з обробкою одного чи декількох рядків матриці A і відповідного елемента вектора b. Кожна ітерація, пов'язана з вирішенням чергової підзадачі, починається з вибору провідного рядка. Шукається рядок з найбільшим за абсолютною величиною значенням серед елементів i-го стовпця, відповідного до змінної xi що виключається.

Оскільки рядки матриці A закріплені за різними підзадачами, для пошуку максимального значення в стовпці підзадачі з номерами k, повинні обмінятися елементами при змінної xi що виключається. Після збору всіх зазначених коефіцієнтів може бути визначено, яка з підзадач містить провідний рядок і яке значення є провідним елементом.

Для продовження обчислень провідна підзадача повинна розіслати свій рядок матриці A і відповідний елемент вектора b всім іншим підзадачам з номерами k, kxi.

У зворотному ході методу Гауса, як тільки будь-яка, наприклад i-та підзадача, визначила свою змінну xi, це значення розповсюджується серед підзадачами з номерами k. У кожній підзадачі отримане невідоме значення множиться на відповідний коефіцієнт і виконується коригування відповідного елемента вектора b.

\\я це перекладала з російської, тому воно трохи дивне. Але це найкраще з того всього що я знайшла.

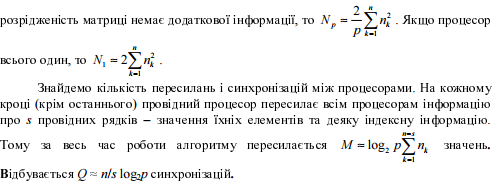
### **4. Які інформаційні взаємодії є між базовими підзадачами для паралельного варіанта методу Гауса?**

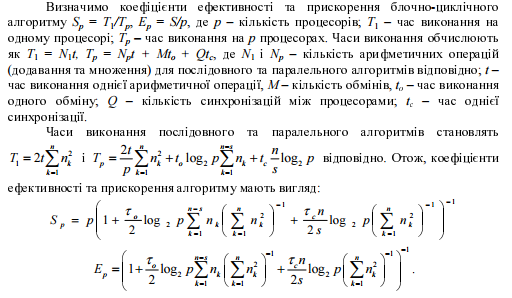
Основним видом інформаційної взаємодії підзадач є операція передачі даних від одного процесора всім процесорам обчислювальної системи.

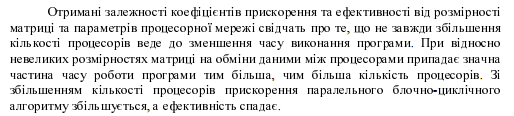
Як результат, для ефективної реалізації необхідних інформаційних взаємодій між базовими підзадачами, топологія мережі передачі даних повинні мати структуру гиперкуба або повного графа.

### **5. Які показники ефективності для паралельного варіанта методу Гауса?**









### **6. У чому полягає схема програмної реалізації паралельного варіанту методу Гауса?**

У розбитті матриці на блоки, щоб можна було виконувати паралельні обчислення над матрицями.

### **7. У чому полягає ідея паралельної реалізації методу сполучених градієнтів?**

Одним з найбільш відомих ітераційних методів є метод сполучених градієнтів, який використовують для розв’язування симетричної додатно-визначеної системи лінійних рівнянь з великою розмірністю.

При розробці паралельного варіанта цього методу в першу чергу потрібно врахувати, що виконання ітераційного методу виконується послідовно, з чого робимо висновок, що найдоцільніше буде розпаралелювати обрахунки, які реалізуються в ході виконання ітерацій.

Головні обрахунки в цьому методі заключаються в перемноженні матриці А на вектори х та d.

Інші обрахунки, що проводяться під час виконаня методу сполучених градієнтів, мають меншу складність. Вони представляють собою різні операції обробки векторів(скалярний добуток, додавання, віднімання, множення на скаляр тощо). Організація таких обрахунків, звісно ж, повинна бути погоджена з паралельним способом виконання операції множення матриці на вектор. Загальні рекомендації можуть бути наступними: при малому розмірі векторів можна дублювати вектори між процесорами. При більшому порядку системи більш доцільно використовувати поділ матриць на блоки.

### **8. Який з алгоритмів (Гауса чи сполучених градієнтів) має більшу комунікаційну складність?**

Складність для градієнтного методу: T_p(comm)=2n(\alpha\cdot\lceil\log p\rceil+w(n/p)(p-1)/ \beta),

де n - розмірність матриці

p - кількість процесорів

\alpha – латентність(біс його знає шо це таке), \beta – пропускна здатність сітки передачі даних, а w розмір елемента впорядковуваних даних в байтах.

Складність для методу Гауса:

T_p^2(comm)=(n-1)\cdot\log_2p\cdot(\alpha+wn/ \beta) ,

### **9. У чому полягає постановка задачі сортування даних?**

Задачу сортування даних можна сформулювати для інформаційної сукупності різної природи - для числової інформації, для слів і символів тексту. Для цього, потрібно визначити поняття порядку для елементів масиву, визначити поняття "більше" і "менше" для кожної пари елементів. Відсортувати послідовність чисел можна таким же способом, як і послідовність рядків тексту. Необхідно тільки визначити який з елементів пари "більший"ніж інший. Більш важливим для вибору алгоритму є місце розташування даних - в оперативній пам'яті комп'ютера або на зовнішньому пристрої.

Тут грає роль відмінність в основних критеріях якості - для даних в оперативній пам'яті основними позитивними властивостями методу є швидкодія і потреби в додатковій пам'яті. Для дискових файлів дуже важливим показником є кількість звернень до пристрою для виконання операцій введення-виведення - воно має бути мінімальним.

Розрізняють два види сортування даних:

* сортування даних, розташованих в оперативній пам'яті комп'ютера (внутрішнє сортування);
* сортування даних, розташованих на зовнішніх запам'ятовуючих пристроях (зовнішнє сортування).

Сформулюємо постановку задачі сортування даних для внутрішнього сортування одновимірного числового масиву за зростанням.

Мається одновимірний масив чисел, що складається з n елементів: X [n]. Переставити елементи масиву так, щоб їх значення розташовувалися в порядку зростання. Іншими словами, для будь-якої пари елементів X [i] і X [i +1] виконується нерівність виду:

Х[i] <= X [i +1].

У цій задачі однозначно визначається структура даних для внутрішнього сортування (одновимірний масив) та порядок упорядкування елементів. Ключем для визначення порядку елементів є числове значення елемента масиву. Результатом розв'язання задачі має бути програма сортування масиву одним або кількома методами. При розробці програми можна скористатися різними алгоритмами

### **10. Наведіть кілька прикладів алгоритмів сортування? Яка обчислювальна складність наведених алгоритмів?**

* [Двійкове дерево пошуку](https://uk.wikipedia.org/wiki/%D0%94%D0%B2%D1%96%D0%B9%D0%BA%D0%BE%D0%B2%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D1%88%D1%83%D0%BA%D1%83)
* [Ниткоподібне сортування](https://uk.wikipedia.org/wiki/%D0%9D%D0%B8%D1%82%D0%BA%D0%BE%D0%BF%D0%BE%D0%B4%D1%96%D0%B1%D0%BD%D0%B5_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)
* [Пірамідальне сортування](https://uk.wikipedia.org/wiki/%D0%9F%D1%96%D1%80%D0%B0%D0%BC%D1%96%D0%B4%D0%B0%D0%BB%D1%8C%D0%BD%D0%B5_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)
* [Плавне сортування](https://uk.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%B2%D0%BD%D0%B5_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)
* [Сортування бульбашкою](https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B1%D1%83%D0%BB%D1%8C%D0%B1%D0%B0%D1%88%D0%BA%D0%BE%D1%8E)
* [Сортування вибором](https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%B8%D0%B1%D0%BE%D1%80%D0%BE%D0%BC)
* [Сортування включенням](https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%BD%D1%8F%D0%BC)
* [Сортування гнома](https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B3%D0%BD%D0%BE%D0%BC%D0%B0)
* [Сортування гребінцем](https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B3%D1%80%D0%B5%D0%B1%D1%96%D0%BD%D1%86%D0%B5%D0%BC)
* [Сортування за розрядами](https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B7%D0%B0_%D1%80%D0%BE%D0%B7%D1%80%D1%8F%D0%B4%D0%B0%D0%BC%D0%B8)
* [Сортування злиттям](https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B7%D0%BB%D0%B8%D1%82%D1%82%D1%8F%D0%BC)
* [Сортування змішуванням](https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B7%D0%BC%D1%96%D1%88%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F%D0%BC)
* [Сортування комірками](https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BA%D0%BE%D0%BC%D1%96%D1%80%D0%BA%D0%B0%D0%BC%D0%B8)
* [Сортування підрахунком](https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D1%96%D0%B4%D1%80%D0%B0%D1%85%D1%83%D0%BD%D0%BA%D0%BE%D0%BC)
* [Сортування порівняннями](https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D0%BE%D1%80%D1%96%D0%B2%D0%BD%D1%8F%D0%BD%D0%BD%D1%8F%D0%BC%D0%B8)
* [Сортування Шелла](https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%A8%D0%B5%D0%BB%D0%BB%D0%B0)
* [Стабільне сортування](https://uk.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%B1%D1%96%D0%BB%D1%8C%D0%BD%D0%B5_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)
* [Топологічне сортування](https://uk.wikipedia.org/wiki/%D0%A2%D0%BE%D0%BF%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%87%D0%BD%D0%B5_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)
* [Цифрове сортування](https://uk.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B5_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)
* [Швидке сортування](https://uk.wikipedia.org/wiki/%D0%A8%D0%B2%D0%B8%D0%B4%D0%BA%D0%B5_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)
* [Odd-even sort](https://uk.wikipedia.org/wiki/Odd-even_sort)

### **11. Яка операція є базовою для задачі сортування даних?**

**Операція порівняння**

Операція порівняння це просто спосіб визначення, який елемент в списку повинен бути на першому місці. Якщо ми сортуємо список чисел від найменшого до найбільшого, операція порівняння говорить нам, щоб помістити число з найменшим значенням першим. Якщо ми сортуємо список букв в алфавітному порядку, операція порівняння говорить нам про те, 'а' перед 'B', 'B' до 'C', і так далі. Ми побачимо, що алгоритм сортування повинен виконати, як правило, багато порівнянь, щоб правильно відсортувати список.

**Операція swap**

Це один із способів як ми переміщаємо елементи при сортуванні. Замінюємо елементи менші у порівнянні з більшими аби помістити всі елементи у правильному порядку. У комп’ютері ця операція може бути складнішою через різні способи копіювання даних.

### **12. У чому суть паралельного узагальнення базової операції задачі сортування даних?**

Для паралельного узагальнення виділеної базової операції сортування розглянемо спочатку ситуацію, коли кількість процесорів збігається з числом сортованих значень (тобто *p = n*). Тоді порівняння значень  *ai* і *aj*, які розташовані, наприклад, на процесорах *Pi* та *Pj*, можна організувати наступним чином:

- виконати взаємообмін наявних на процесорах *Pi* та *Pj* значень (зі збереженням на цих процесорах початкових елементів);

- порівняти на кожному процесорі *Pi* та *Pj* отримані однакові пари значень (*ai, aj*); результати порівняння використовуються для розділення даних між процесорами - на одному процесорі (наприклад, *Pi* ) залишається менший елемент, інший процесор (тобто *Pj*) запам'ятовує для подальшої обробки більшого значення пари:

*ai' =* min (*ai, aj*), *aj' =* max (*ai, aj*).

Розглянуте паралельне узагальнення базової операції сортування може бути належним чином адаптовано і для випадку p < n, коли кількість процесорів є меншим числа впорядковуємих значень. У даній ситуації кожен процесор буде містити вже не єдине значення, а частину (блок розміром n/p) сортованого набору даних. Ці блоки зазвичай упорядковуються на самому початку сортування на кожному процесорі окремо за допомогою якого-небудь швидкого алгоритму (попередня стадія паралельного сортування). Далі, за схемою одноелементного порівняння, взаємодія пари процесорів Pi та Pj для спільного упорядкування вмісту блоків *Ai* та *AJ*  може бути здійснено наступним чином:

- виконати взаємообмін блоків між процесорами *Pi* та *Pj* ;

- об'єднати блоки *Ai* та *AJ*  на кожному процесорі в один відсортований блок подвійного розміру (при початковій впорядкованості блоків *Ai* та *AJ*  процедура їх об'єднання зводиться до швидкої операції злиття впорядкованих наборів даних);

-  розділити отриманий подвійний блок на дві рівні частини і залишити одну з цих частин (наприклад, з меншими значеннями даних) на процесорі *Pi*, а другу частину (з великими значеннями відповідно) - на процесорі *Pj*

[*Ai  AJ* ]*сорт* = *Ai' AJ'* ; ai' *Ai'*, ; aJ' *AJ'*  ai' aj'

Розглянута процедура іменується в літературі як операція "порівняти і розділити" (compare-split). Слід зазначити, що сформовані в результаті такої процедури блоки на процесорах *Pi*  та *Pj* збігаються за розміром з початковими блоками *Ai* та *AJ*  і всі значення , які розташовані на процесорі, являються меншими значень на процесорі *Pj*

### **13. Що являє собою алгоритм парно-непарної перестановки?**

**Odd-even sort** чи **Odd-even transposition sort** — в інформатиці, парне-непарне сортування (також відоме як сортування цеглинами) є відносно простим алгоритмом сортування, розробленим спочатку для використання на паралельних процесорах з локальними взаємозв'язками. Воно порівнюється з сортуванням бульбашкою, з яким він поділяє багато характеристик. Алгоритм діє наступним чином: порівнюються всі парні / непарні пари проіндексованих суміжних елементів в списку, і якщо пара знаходиться в неправильному порядку (перший більше, ніж другий) елементи міняються місцями. Наступним кроком повторює це для парних / непарних індексованих пар (суміжних елементів). Чергуються парні/непарні та непарні/парні кроки, поки список не буде відсортований.

### **14. У чому полягає паралельний варіант алгоритму Шелла? Які основні відмінності цього варіанту паралельного алгоритму сортування від методу парно-непарної перестановки?**

Для алгоритму Шелла може бути запропонований паралельний аналог методу, якщо топологія комунікаційний мережі має структуру N -мірного гіперкуба (тобто кількість процесорів дорівнює ). Виконання сортування в такому випадку може бути розділене на два послідовних етапи. На першому етапі (N ітерацій) здійснюється взаємодія процесорів, які є сусідніми в структурі гіперкуба (але ці процесори можуть виявитися далекими при лінійної нумерації). Другий етап полягає в реалізації звичайних ітерацій паралельного алгоритму парно-непарної перестановки. Ітерації даного етапу виконуються до припинення фактичної зміни набору, який сортується, тож, загальна кількість таких ітерацій може бути різним - від 2 до . Трудомісткість паралельного варіанта алгоритму Шелла визначається виразом:

,

де друга і третя частини співвідношення фіксують обчислювальну складність першого і другого етапів сортування відповідно. Як можна помітити, ефективність даного паралельного способу сортування виявляється краще показників звичайного алгоритму парно-непарної перестановки при .

\\знову ж таки, переклад з російської… -\_-

### **15. Що являє собою паралельний варіант алгоритму швидкого сортування?**

Паралельне узагальнення алгоритму швидкого сортування найбільш простим способом може бути отримано для обчислювальної системи з топологією у вигляді

*N*-мірного гіперкуба (тобто *p=2N*). Нехай, як і раніше, початковий набір даних розподілений між процесорами блоками однакового розміру *n/p*; результуюче розташування блоків має відповідати нумерації процесорів гіперкуба. Можливий спосіб виконання першої ітерації паралельного методу при таких умовах може полягати в наступному:

- вибрати будь-яким чином провідний елемент і розіслати його по всім процесорам системи;

- розділити на кожному процесорі чинний блок даних на дві частини з використанням отриманого провідного елементу;

- утворити пари процесорів, для яких бітове подання номерів відрізняється тільки в позиції *N*, і здійснити взаємообмін даними між цими процесорами; в результаті таких пересилань даних на процесорах, для яких у бітовому представленні номера біт позиції *N* дорівнює 0, повинні виявитися частини блоків зі значеннями, меншими провідного елементу; процесори з номерами, в яких біт *N* дорівнює 1, повинні зібрати, відповідно, всі значення даних, що перевищують значення провідного елементу.

У результаті виконання такої ітерації сортування початковий набір виявляється розділеним на дві частини, одна з яких (зі значеннями меншими, ніж значення провідного елемента) розташовується на процесорах, у бітовому представленні номерів яких біт *N* дорівнює 0. Таких процесорів всього *p/2*, таким чином, початковий *N*-мірний гіперкуб також виявляється розділеним на два гіперкуба розмірності *N-1*. До цих підгіперкубів, у свою чергу, може бути паралельно застосована описана вище процедура. Після *N* -кратного повторення подібних ітерацій для завершення сортування досить впорядкувати блоки даних, які отримані на кожному окремому процесорі обчислювальної системи. Ефективність паралельного методу швидкого сортування, як і у послідовному варіанті, багато в чому залежить від успішності вибору значень провідних елементів.

Тривалість виконуваних операцій передачі даних визначається операцією розсилки провідного елементу на кожній ітерації сортування - загальна кількість міжпроцесорних обмінів для цієї операції на *N*-мірному гіперкубі може бути обмежена оцінкою

log2*p* ,

і взаємообміном частин блоків між сусідніми парами процесорів - загальна кількість таких передач збігається з кількістю ітерацій сортування, тобто рівне log p, обсяг переданих даних не перевищує подвоєного обсягу процесорного блоку, тобто обмежений величиною 2^n/p.

Обчислювальна трудомісткість методу обумовлюється складністю локального сортування процесорних блоків, часом вибору провідних елементів і складністю поділу блоків, що в цілому може бути виражене за допомогою співвідношення:

*T1=(n/p)*log*(n/p)+*log *p+*log*(n/p)*log *p*

(при побудові даної оцінки передбачалося, що для вибору значення провідного елемента при впорядкованості процесорних блоків даних достатньо однієї операції).

\\ну я намагалась...

### **16. Що залежить від правильного вибору провідного елементу для паралельного алгоритму швидкого сортування?**

Вибір провідного елемента впливає на складність алгоритму, тобто в загальному випадку вибір мінімального або максимального як провідного дає найгіршу швидкодію.

При виборі базового елементу, в ідеалі повинен вибиратися середній зі списку. Тоді при розбитті він розділиться на дві рівні половини. Тільки обчислити середнє значення у списку дуже складно, тому навіть найшвидша сортування обходить це числення стороною. Але і вибір основного елемента з максимальним або мінімальним значенням – також не найкращий варіант. У разі такого визначення один із створених списків буде гарантовано порожній, а другий переповнений. Звідси висновок, що в якості базового елементу слід вибирати той, який знаходиться ближче до середнього, але далі від максимального і мінімального.

### **17. Які способи вибору провідного елементу можуть бути запропоновані?**

Встановимо два індекси на 1-й (індекс *і*) і на останній (індекс *j*) елементи послідовності. Потім, поки елемент з індексом *j* менший або дорівнює елементу з індексом *і*, будемо зменшувати *j* на 1. Якщо ж елемент з індексом *j* більший або дорівнює елементу з індексом *і*, то міняємо місцями елементи з індексами *і* та *j*. Потім, поки елемент з індексом *j* менший або дорівнює елементу з індексом *і*, будемо збільшувати *і* на 1. Якщо ж елемент з індексом *j* більший або дорівнює елементу з індексом *і*, то міняємо місцями елементи з індексами *і* та *j*.

Цей процес продовжується доти, поки *j* не стане дорівнювати *і*. Елемент з індексом *і = j* і буде шуканим

### **18. Для яких топологій можуть застосовуватися алгоритми сортування: Шелла, швидке, парно-непарної перестановки?**

Шелла - гіперкуб  
Швидке сортування - гіперкуб

// парно-непарної не знайшла, але коротше гіперкуб… (н-мірний)

### **19. У чому полягає алгоритм сортування з використанням регулярного набору зразків?**

Алгоритм сортування з використанням регулярного набору зразків (the parallel sorting by regular sampling) також є узагальненням методу швидкого сортування.

Алгоритм:

1. упорядкування наявних на процесорах блоків. Дана операція може бути виконана кожним процесором незалежно один від одного за допомогою звичайного алгоритму швидкого сортування; далі кожен процесор формує набір з елементів своїх блоків з індексами 0, m, 2m, ..., (p-1) m, де m = n / p^2;

2.всі сформовані на процесорах набори даних збираються на одному з процесорів системи і об'єднуються в ході послідовного злиття в одну впорядковану множину. Далі з отриманого безлічі значень з елементів з індексами  формується новий набір провідних елементів, який передається всім використовуваним процесорам. На завершення етапу кожен процесор виконує поділ свого блоку на p частин з використанням отриманого набору провідних значень;

3. кожен процесор здійснює розсилку виділених раніше частин свого блоку всім іншим процесорам системи; розсилка виконується відповідно до порядку нумерації - частина j, 0 <= j <p, кожного блоку пересилається процесору з номером j;

4. на четвертому етапі виконання алгоритму кожен процесор виконує злиття p отриманих частин в один відсортований блок.

По завершенні четвертого етапу вихідний набір даних стає відсортованим.

### **20. Наведіть визначення графа. Які основні способи використовуються для задання графів?**

Граф — це множина X і множина відношень R, заданих на X. Графічно множина X зображається точками, що називаються вершинами графа, а множина R — лініями, так званими ребрами (дугами) графа.

Якщо граф має багато вершин і ребер, то він втрачає оглядовість. З таким графом важко працювати: у підрахунках трапляються помилки. Тоді відображають граф у вигляді матриці М. Найбільш поширеними є такі матриці графів: матриця суміжності відповідно вершин і ребер, матриця інциденті, матриця доступності, матриця відстаней.

Матриця суміжності вершин графа - це квадратна матриця, в якій рядки і стовпчики відповідають вершинам графа. Порядок матриці збігається з кількістю вершин. При заповненні матриці на перетині рядка і стовпчика, що відповідають суміжним вершинам, ставлять 1; решту комірок заповняють нулями.

Матриця інциденції дуг графа – це ортогональна матриця, в якій стовпці відповідають вершинам графа, а рядки – ребрам. Матрицю заповнюють так: на перетині стовпчика, що відповідає певній вершині, з рядком, що відповідає ребру, яке з цієї вершини виходить, ставлять -1; з рядком, що відповідає ребру, яке в цю вершину входить, ставлять +1; з рядком, що відповідає ребру, яке неінцидентне даній вершині – 0.

Матриці суміжності та інциденції визначають графи з точністю до ізоморфізму. За цими матрицями можна точно накреслити ізоморфні їм графи.

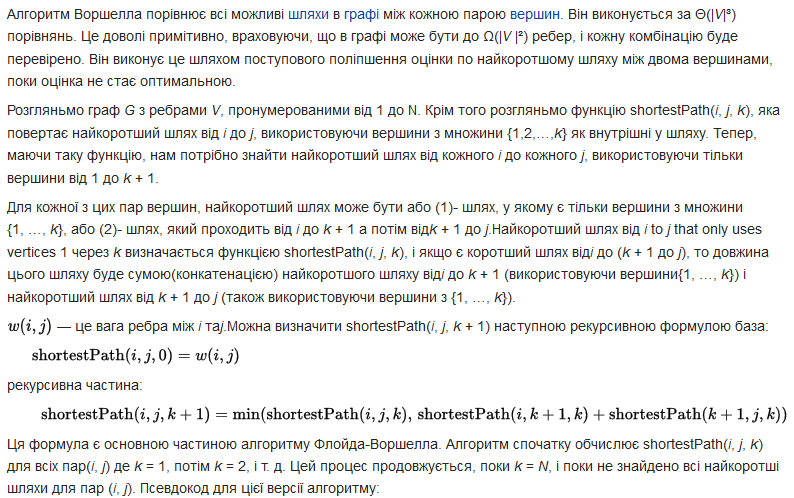
### **21. У чому полягає завдання пошуку всіх найкоротших шляхів?**

**Задача про найкоротші шляхи для всіх пар**, тут ми маємо знайти найкоротші шляхи між кожною парою вершин *v*, *v'* в графі.

//не знаю що він ще захоче сюди, але писати ще щось - сенсу?

### **22. Наведіть загальну схему алгоритму Флойда. Яка трудомісткість алгоритму?**

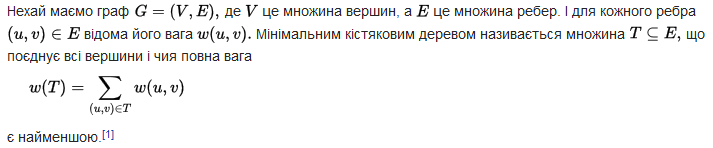
В інформатиці, **алгоритм Флойда-Воршелла** використовується для розв'язання задачі про найкоротший шлях у зваженому графі з додатними або від'ємними вагами ребер (але без від'ємнозначних циклів). При звичайній реалізації алгоритм видасть довжини (сумарні ваги) найкоротших шляхів між *всіма* парами вершин, хоча він не видасть інформацію про самі шляхи.



Нехай |V|= *n*- кількість вершин. Щоб знайти усі *n*2 з shortestPath(*i*,*j*,*k*) (для всіх *i* та *j*) з даного shortestPath(*i*,*j*,*k*−1) потрібно 2*n*2 операцій. Ми починаємо з shortestPath(*i*,*j*,0) = edgeCost(*i*,*j*) рахуємо послідовність *n* матриць shortestPath(*i*,*j*,1), shortestPath(*i*,*j*,2), …, shortestPath(*i*,*j*,*n*), кількість операцій = *n* · 2*n*2 = 2*n*3 тому складність алгоритму = Θ(*n*3).

### **23. У чому полягає спосіб розпаралелювання алгоритму Флойда?**

### **24. У чому полягає завдання знаходження мінімального шляху дерева? Наведіть приклад використання завдання на практиці.**



* Розпізнавання математичних виразів (від руки)
* Таксономія - наука класифікації
* еКластерний аналіз
* Комп’ютерні мережі - оптимальні з’єднання
* Проектування плат, схем

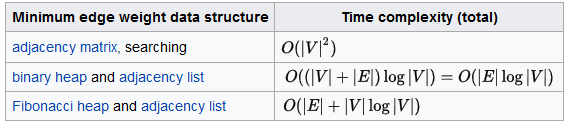
### **25. Наведіть загальну схему а лгоритму Прима. Яка трудомісткість алгоритму?**

**Алгоритм Прима** - алгоритм побудови мінімального кістякового дерева зваженого зв'язного неорієнтованого графа. Це жадібний алгоритм.

Побудова починається з дерева, що включає в себе одну (довільну) вершину. Протягом роботи алгоритму дерево розростається, поки не охопить всі вершини вихідного графа. На кожному кроці алгоритму до поточного дереву приєднується найлегше з ребер, що з'єднують вершину з побудованого дерева і вершину, що не належить дереву.

Алгоритм

1. Спочатку ребра сортують за зростанням ваги.
2. Додають найменше ребро в дерево.
3. Зі списку ребер із найменшою вагою вибирають таке нове ребро, щоб одна з його вершин належала дереву, а інша — ні.
4. Це ребро додають у дерево і знову переходять до кроку 3.
5. Робота закінчується, коли всі вершини будуть у дереві.



### **26. У чому полягає спосіб розпаралелювання алгоритму Прима?**

Спосіб полягає в розпаралеленні пошуку мінімального ребра, ще не включеного в каркас.

U: = {a}, де а - довільна вершина графа

F: = "Порожня множина"

W: = V - {a}

while W не пусте do

1. на кожному процесі знайти ребро найменшої ваги ej = (x, y) серед всіх таких ребер, у яких х належить U, а y належить Vj і W
2. зібрати всі ej на процесі з рангом 0
3. на процесі з рангом 0 вибрати ребро e, мінімальне з усіх ej
4. на процесі з рангом 0 F: = F + {e}
5. на процесі з рангом 0 U: = U + {y}
6. на процесі з рангом 0 W: = W - {y}
7. оновити на всіх процесах множину U

### **27. У чому відмінність геометричних і комбінаторних методів розділення графа? Які методи є кращими? Чому?**

Геометричні методи:

* Геометричні методи виконують розбиття мереж, грунтуючись виключно на координатної інформації про вузли мережі,
* Так як геометричні методи не беруть до уваги інформацію про зв'язності елементів мережі, то вони не можуть явно призвести до мінімізації сумарної ваги граничних ребер. Для мінімізації міжпроцесорних комунікацій геометричні методи оптимізують деякі допоміжні показники (наприклад, довжину границі між розділеними ділянками мережі),
* Геометричні методи працюють виключно швидко,
* Геометричні методи, що розглядаються(Покоординатно розбиття, Рекурсивний інерційний метод поділу навпіл, Розподіл мережі з використанням кривих Пеано)

Комбінаторні методи:

* На відміну від геометричних методів, комбінаторні алгоритми зазвичай оперують не з мережею, а з графом, побудованим для цієї мережі,
* Комбінаторні методи не беруть до уваги інформацію про близькість розташування елементів мережі один відносно іншого, керуючись тільки суміжністю вершин графа,
* Комбінаторні методи забезпечують більш збалансоване розбиття і менше інформаційну взаємодію отриманих підмереж в порівнянні з геометричними методами
* Час роботи комбінаторних методів, як правило, істотно перевершує часи роботи геометричних,
* Розглянуті геометричні методи (Розподіл з урахуванням зв'язності, Алгоритм Керніган-Ліна)

### **28. Наведіть опис методу покоординатного розбиття і алгоритму поділу з урахуванням зв'язності. Який з цих методів є більш простим для реалізації?**

**Покоординатне розбиття:**

Загальна схема виконання методу:

1. Обчислюються центри мас елементів мережі,
2. Отримані точки проектуються на вісь, ту що відповідає найбільшій стороні розділяючої мережі.

**Поділ з урахуванням зв’язності:**

На кожній ітерації алгоритму відбувається поділ графа на 2 частини. Поділ графа на необхідне число частин досягається шляхом рекурсивного застосування алгоритму.

Загальна схема алгоритму:

1. Iteration = 0

2. Привласнення номера Iteration довільній вершині графа

3. Присвиразувоєння ненумерований сусідам вершин з номером Iteration номера Iteration + 1

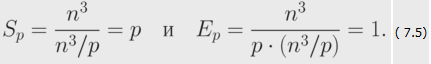
4. Iteration = Iteration + 1

5. Якщо ще є неперенумерованние сусіди, то перехід на крок 3

6. Поділ графа на 2 частини в порядку нумерації

### **29. Проведіть аналіз і отримаєте показники ефективності стрічкового алгоритму при горизонтальному розбитті перемножуваних матриць.**

Для паралельного алгоритму на кожній ітерації кожен процесор виконує множення наявних на процесорі смуг матриці А і матриці В (розмір смуг дорівнює n / p, і, як результат, загальна кількість виконуваних при цьому множенні операцій одно n3 / p2). Оскільки число ітерацій алгоритму збігається з кількістю процесорів, складність паралельного алгоритму без урахування витрат на передачу даних може бути визначена за допомогою 

З урахуванням цієї оцінки показники прискорення і ефективності даного паралельного алгоритму матричного множення приймають вид:

Таким чином, загальний аналіз складності дає ідеальні показники ефективності паралельних обчислень. Для уточнення отриманих співвідношень оцінимо більш точно кількість обчислювальних операцій алгоритму і врахуємо витрати на виконання операцій передачі даних між процесорами.

З урахуванням числа і тривалості виконуваних операцій час виконання обчислень паралельного алгоритму може бути оцінений наступним чином:



(Тут, як і раніше, tau є час виконання однієї елементарної скалярною операції).

Для оцінки комунікаційної складності паралельних обчислень будемо припускати, що всі операції передачі даних між процесорами в ході однієї ітерації алгоритму можуть бути виконані паралельно. Обсяг переданих даних між процесорами визначається розміром смуг і становить n / p рядків або стовпців довжини n. Загальна кількість паралельних операцій передачі повідомлень на одиницю менше числа ітерацій алгоритму (на останній ітерації передача даних не є обов'язковою). Тим самим, оцінка трудомісткості виконуваних операцій передачі даних може бути визначена як



де \ alpha - латентність, \ beta - пропускна здатність мережі передачі даних, а w є розмір елемента матриці в байтах.

З урахуванням отриманих співвідношень загальний час виконання паралельного алгоритму матричного множення визначається наступним виразом:



### **30. Які інформаційні взаємодії виконуються для алгоритмів при стрічковій схемі розділення даних (множення матриць)?**

Алгоритм - ітераційна процедура, кількість ітерацій якої збігається з числом підзадач. На кожній ітерації алгоритму кожна підзадача містить по одному рядку матриці А і одному стовпцю матриці В. При виконанні ітерації проводиться скалярне множення рядків і стовпців в підзадачі, що призводить до отримання відповідних елементів результуючої матриці С. Після закінчення обчислень в кінці кожної ітерації стовпці матриці В повинні бути передані між підзадачами так, щоб в кожній підзадачі виявилися нові стовпці матриці в і могли бути обчислені нові елементи матриці C. При цьому дана передача стовпців між підзадачами повинна бути організована таким чином, щоб після завершення ітерацій алгоритму в кожній підзадачі послідовно виявилися всі стовпці матриці В.

Можлива проста схема організації необхідної послідовності передач стовпців матриці В між підзадачами полягає в поданні топології інформаційних зв'язків підзадач у вигляді кільцевої структури. У цьому випадку на кожній ітерації підзадача i, 0 <= i <n, буде передавати свій стовпець матриці В підзадачі з номером i + 1 (відповідно до кільцевої структури підзадача n-1 передає свої дані підзадаче з номером 0). Після виконання всіх ітерацій алгоритму необхідна умова буде забезпечена - в кожній підзадачі по черзі виявляться всі стовпці матриці В.

### **31. Які інформаційні взаємодії виконуються для блокових алгоритмів множення матриць?**

Отже, за основу паралельних обчислень для матричного множення при блочному поділі даних прийнятий підхід, при якому базові підзадачі відповідають за обчислення окремих блоків матриці C і при цьому в підзадача на кожній ітерації розрахунків розташовується тільки по одному блоку вихідних матриць A і B. Для нумерації підзадач використовуватимемо індекси розміщуються в підзадача блоків матриці C, тобто підзадача (i, j) відповідає за обчислення блоку Cij - тим самим, набір підзадач утворює квадратну решітку, відповідну структурі блочного подання матриці C.

Можливий спосіб організації обчислень при таких умовах полягає в застосуванні широко відомого алгоритму Фокса (Fox).

Відповідно до алгоритму Фокса в ході обчислень на кожній базовій підзадачі (i, j) розташовується чотири матричних блоки:

* блок Cij матриці C, який вираховується підзадачею;
* блок Aij матриці A, що розміщується в підзадачі перед початком обчислень;
* блоки A'ij, B'ij матриць A і B, одержувані підзадачею в ході виконання обчислень.

Виконання паралельного методу включає:

* етап ініціалізації, на якому кожній підзадачі (i, j) передаються блоки Aij, Bij і обнуляються блоки Cij на всіх підзадач;
* етап обчислень, в рамках якого на кожній ітерації l, 0 <= l <q, здійснюються такі операції:
  + для кожного рядка i, 0 <= i <q, блок Aij підзадачі (i, j) пересилається на всі підзадачі тієї ж рядки i решітки; індекс j, що визначає положення підзадачі в рядку, обчислюється відповідно до вираження 
  + отримані в результаті пересилань блоки A'ij, B'ij кожної підзадачі (i, j) перемножуються і додаються до блоку Cij
  + блоки B'ij кожної підзадачі (i, j) пересилаються підзадачам, що є сусідами зверху в стовпцях решітки підзадач (блоки підзадач з першого рядка решітки пересилаються підзадач останнього рядка решітки).

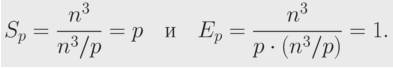
### **32. Яка топологія комунікаційної мережі є доцільною для кожного з розглянутих алгоритмів множення матриць?**

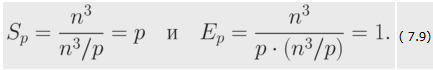
Алгоритми, засновані на стрічковому розподілі даних, орієнтовані на топологію мережі у вигляді гіперкуба або повного графа. Для реалізації алгоритмів, заснованих на блочному поділі даних, необхідна наявність топології решітки.

### **33. Який з розглянутих алгоритмів множення матриць характеризується найменшими і найбільшими вимогами до обсягу необхідної пам'яті?**

//Алгоритм Штрассена. Недоліком даного методу є велика складність програмування в порівнянні зі стандартним алгоритмом, слабка чисельна стійкість і більший обсяг використовуваної пам'яті.

### **34. Який з розглянутих алгоритмів множення матриць володіє найкращими показниками прискорення і ефективності?**

Стрічковий:

Блочний:

//різниці бляха ж немає

### **35. Оцініть можливість виконання матричного множення як послідовності операцій множення матриці на вектор.**

Для багатьох методів матричних обчислень характерним є повторення одних і тих же операцій для різних даних. Дана властивість свідчить про наявність паралелізму за даними при виконанні матричних обчислень, і, як результат, розпаралелювання матричних операцій зводиться, в більшості випадків, до розбиття оброблюваних матриць між процесорами використовуваної обчислювальної системи.

//не все

**36. Дайте загальну характеристику програмної реалізації алгоритму Фокса. У чому можуть полягати відмінності в програмній реалізації інших розглянутих алгоритмів?**

- кожен з процесорів решітки відповідає за обчислення одного блоку матриці C;

- в ході обчислень на кожному з процесорів розташовується чотири матричні блоки:

o блок матриці С, обчислюваний процесором;

o блок матриці A, розміщений в процесорі перед початком обчислень;

o блоки матриць А і В, отримувані процесором в ході виконання обчислень;

Виконання паралельного методу включає:

- **етап ініціалізації**, на якому на кожен процесорів передаються блоки , і обнуляються блоки на всіх процесорах;

- **етап обчислень**, на кожній ітерації , якого виконується:

o для кожного рядка , процесорної решітки блок процесора пересилається на всі процесори того ж рядка *i*; індекс *j*, який визначає положення процесора в рядку, обчислюється за співвідношенням

( mod є операція отримання остачі від цілого ділення);

o отримані в результаті пересилань блоки кожного процесора перемножуються і додаються до блоку

o блоки кожного процесора пересилаються процесорам , які є сусідами зверху в стовпцях процесорної решітки.

### **37. У чому полягає важливість стандартизації засобів передачі повідомлень?**

* MPI дозволяє в значній мірі знизити гостроту проблеми переносимості паралельних програм між різними комп'ютерними системами - паралельна програма, розроблена на алгоритмічній мові C або Fortran з використанням бібліотеки MPI, як правило, буде працювати на різних обчислювальних платформах;
* MPI сприяє підвищенню ефективності паралельних обчислень, оскільки в даний час практично для кожного типу обчислювальних систем існують реалізації бібліотек MPI, в максимальному ступені враховують можливості комп'ютерного обладнання;
* MPI зменшує, в певному плані, складність розробки паралельних програм, бо, з одного боку, велика частина основних операцій передачі даних передбачена стандартом MPI, а з іншого боку, вже є велика кількість бібліотек паралельних методів, створених з використанням MPI.

### **38. Що слід розуміти під паралельною програмою?**

Під паралельною програмою в рамках МРІ розуміють множину одночасно виконуваних процесів. Процеси можуть виконуватися на різних процесорах, але на одному процесорі можуть розташовуватися і декілька процесів (в цьому випадку їх виконання здійснюється в режимі розділення часу). В граничному випадку для виконання паралельної програми може використовуватися один процесор - як правило, такий спосіб застосовується для початкової перевірки правильності паралельної програми. Кожний процес програми породжується на основі копії одного і того ж програмного коду (модель SPMP). Цей програмний код, зображуваний у вигляді виконуваної програми, повинен бути доступним в момент запуску паралельної програми на всіх використовуваних процесорах. Вихідний програмний код для виконуваної програми розроблюється на алгоритмічних мовах С чи [FORTRAN](https://uk.wikipedia.org/wiki/FORTRAN) із застосуванням тієї чи іншої реалізації бібліотеки МРІ. Кількість процесів та чисельність використовуваних процесорів визначається в момент запуску паралельної програми засобами середовища виконання МРІ - програм і в ході обчислень не може змінюватися без застосування спеціальних, але рідко задіяних засобів динамічного породження процесів та управління ними, які з’явилися в стандарті МРІ версії 2.0. Всі процеси програми послідовно перенумеровані від ) до , де є загальна кількість процесорів. Номер процесу іменується рангом процесу.

### **39. У чому відмінність понять процесу і процесора?**

**Проце́сор**– основний компонент комп'ютера, призначений для керування всіма його пристроями та виконання арифметичних і логічних операцій над даними.

**Процес** — об'єкт операційної системи, контейнер системних ресурсів, призначених для підтримки виконання програми, це контейнер для ресурсів, які використовуються при виконанні екземпляра програми.

### **40. Який мінімальний набір функцій MPI дозволяє почати розробку паралельних програм?**

Першою функцією МРІ, що викликається, повинна бути функція:

int MPI\_Init(int \*argc, char \*\*\*argv)

де - argc - вказівник на кількість параметрів командної стрічки,

- argv - параметри командної стрічки,

яка застосовується для ініціалізації середовища виконання МРІ - програми. Параметрами функції є кількість аргументів в командній стрічці та адреса вказівника на масив символів тексту самої командної стрічки. Останньою функцією МРІ, що викликається, обов’язково повинна бути функція:

int MPI\_Finalize (void)

### **41. Як описуються передані повідомлення?**

Для передачі повідомлень-відправник повинен виконати функцію:

int MPI\_Send(void \*buf, int count, MPI\_Datatype type, int dest, int tag, MPI\_Comm comm),

де

- buf - адреса буфера пам’яті, в якому розташовані дані відправленого повідомлення;

- count - кількість елементів даних повідомлення;

- type - тип елементів даних повідомлення, що пересилається;

- dest - ранг процесів, якому відправляється повідомлення;

- tag - значення-тег, яке використовується для ідентифікації повідомлення;

- comm - комунікатор, в рамках якого виконується передача даних.[[3]](https://uk.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%BB%D0%B5%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BD%D0%B0_%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D1%96_%D0%9C%D0%A0%D0%86#cite_note-3)

### **42. Які основні етапи проектування та розробки методів паралельних обчислень?**

На стадії проектування паралельний метод може бути представлений у вигляді графу «підзадачі-повідомлення» (агреговане представлення графу «операції-операнди»).

Модель «підзадачі-повідомлення» дозволяє:

• Визначити підзадачі однакової обчислювальної складності.

• Забезпечити низький рівень інформаційної залежності між підзадачами.

Для розробки паралельних алгоритмів необхідно виконати:

• виділення підзадач;

• визначення інформаційних залежностей;

• масштабування;

• розподіл підзадач по процесорам (процесам) обчислювальної системи.

### **43. Як визначається модель "підзадачі-повідомлення"?**

Оптимальне вирішення проблеми розподілу підзадач між обчислювальними елементами грунтується на аналізі інформаційної зв’язності графу «підзадачі – повідомлення». Так, зокрема, підзадачі, між якими є інформаційні взаємодії, доцільно розміщувати на обчислювальних елементах, між якими існують прямі лінії передачі даних. Слід зауважити, що вимога мінімізації інформаційних обмінів між обчислювальними елементами може суперечити умові рівномірного завантаження процесів. Існує можливість розміщення всіх підзадач на одному обчислювальному елементі та повне усунення передачі повідомлень, проте, зрозуміло, що завантаження більшості обчислювальних елементів у цьому випадку буде мінімальною.

Модель «підзадачі-повідомлення» дозволяє:

• Визначити підзадачі однакової обчислювальної складності.

• Забезпечити низький рівень інформаційної залежності між підзадачами.

### **44. Як визначається модель "процеси-канали"?**

В цій моделі програми складаються з одного та більше процесів, розподілених по процесорах. Процеси виконуються одночасно, їх кількість може змінюватись на протязі часу виконання програми. Процеси обмінюються даними через канали, які представляють собою однонаправлені комунікаційні лінії, які з’єднують тільки два процеси. Канали можна створювати та видаляти.

Модель процес\канал характеризується наступними властивостями:

1. Паралельне обчислення складається з одного та більше одночасно виконуваних процесів, число яких може змінюватись під час виконування програми

2. Процес – це послідовна програма з локальними даними. Процес має вхідні та вихідні порти, які служать інтерфейсом до середовища процесу.

3. В додаток до звичайних операцій процес може виконувати наступні дії: Надіслати повідомлення через вихідний порт, отримати повідомлення з вхідного порту, створити новий процес та завершити процес.

4. Операція відправлення асинхронна – вона завершується відразу, не очікуючи того, коли данні будуть отримані. Операція отримання синхронна – вона блокує процес до моменту отримання повідомлення.

5. Пари з вхідного та вихідного портів з’єднуються чергами повідомлень, які називаються каналами. Канали можна створювати та видаляти. Посилання на канали можна включати в повідомлення, так що зв’язність може змінюватись динамічно.

6. Процеси можна розподілю вати по фізичним процесорам різними способами, при чому відображення (розподілення) не впливає на семантику програми.

Поняття процесу дозволяє говорити про місцезнаходження даних : дані, які зберігаються в локальній пам’яті процесу – розміщуються «близько», інші дані «віддалені». Поняття каналу забезпечує механізм для вказування того, зо для того, щоб продовжити обчислення одному процесу необхідні дані іншого процесу (залежність по даним).

### **45. Які основні вимоги повинні бути забезпечені при розробці паралельних алгоритмів?**

Розробка алгоритмів паралельних обчислень є складною науково-технічної завданням, оскільки крім чисто математичних питань побудови алгоритмів і дослідження їх збіжності слід обов'язково враховувати архітектуру обчислювального пристрою, для якого розробляється той чи інший алгоритм. Природно, що є деякі базові вимоги до розробки алгоритмів.

Обов'язкові кроки в організації паралельних обчислень

1. Виконання аналізу наявних обчислювальних схем і поділ їх на частини (підзадачі), які можуть бути в значній мірі реалізовані незалежно одне від одного;
2. Пошук для сформованого набору підзадач інформаційних взаємодій, які мають бути задіяні в ході обчислювальних дій;
3. Масштабування підзадач.Властивість масштабованості полягає в ефективному використанні всіх наявних обчислювальних ресурсів.(Паралельне застосування повинне бути готове до того, що завтра воно буде запускатися на системі з великими обчислювальними можливостями).
4. Визначення необхідної для вирішення повної задачі обчислювалюваної системи і провести розподіл набору підзадач між процесорами системи.

### **46. У чому полягають основні дії на етапі виділення підзадач?**

Як правило, кожен процесор виділяється для вирішення певної підзадачі, однак якщо підзадач більше, ніж процесорів, то на один процесор слід запускати кілька підзадач, дотримуючись по можливості збалансованість загальної завантаження. Така ситуація виникає найчастіше на початковому етапі розпаралелювання, коли працездатність модельного алгоритму перевіряють на одному процесорі в режимі поділу часу.

### **47. Які основні дії на етапі визначення інформаційних залежностей?**

При проведенні аналізу інформаційних залежностей між підзадачами слід розрізняти наступні форми інформаційної взаємодії:

- Локальні та глобальні схеми передачі даних. Для локальних схем передачі даних в кожний момент часу виконуються тільки між невеликим числом підзадач, як правило, на сусідніх обчислювальних елементах, для глобальних операцій передачі даних у процесі комунікації беруть участь всі підзадачі.

- Структурні та довільні способи взаємодії. Для структурних способів організація взаємодій призводить до формування деяких стандартних схем комунікації, наприклад, у вигляді кільця, прямокутної решітки та інші, для довільних структур взаємодії схема виконуваних операцій передач даних не носить характер однорідності.

- Статичні або динамічні схеми передачі даних. Для статичних схем моменти та учасники інформаційної взаємодії фіксуються на етапах проектування та розробки паралельних програм, для динамічного варіанта взаємодії структура операції передачі даних визначається в ході виконуваних обчислень.

- Синхронні та асинхронні способи взаємодії. Для синхронних способів операції передачі даних виконуються тільки при готовності всіх учасників взаємодії та завершуються тільки після повного закінчення всіх комунікаційних дій, при асинхронному виконанні операцій учасники взаємодії можуть не чекати повного завершення дій з передачі даних. Для представлених способів взаємодії досить складно виділити необхідні форми організації передачі даних: синхронний варіант, як правило, більш простий для використання, в той час як асинхронний спосіб часто дозволяє істотно знизити часові затримки, викликані операціями інформаційної взаємодії.

### **48. У чому полягають основні дії на етапі масштабування наявного набору підзадач?**

Масштабування розробленої обчислювальної схеми паралельних обчислень проводиться у разі, якщо кількість наявних підзадач відрізняється від числа запланованих до використання обчислювальних елементів. Для скорочення кількості підзадач необхідно виконати агрегацію обчислень. Правила, які застосовуються при реалізації даного етапу співпадають з рекомендаціями початкового етапу виділення підзадач – підзадачі повинні мати однакову обчислювальну складність, а обсяг та інтенсивність інформаційних взаємодій між підзадачами повинні залишатися на мінімально-можливому рівні. Як результат, першими претендентами на об’єднання є підзадачі з високим ступенем інформаційної взаємозалежності. При недостатній кількості наявного набору підзадач для завантаження всіх доступних до використання обчислювальних елементів необхідно виконати деталізацію (декомпозицію) обчислень. Як правило, проведення подібної декомпозиції не викликає яких-небудь ускладнень, якщо для базових завдань методи паралельних обчислень є відомими.

Виконання етапу масштабування обчислень має зводитися до розробки правил агрегації та декомпозиції підзадач, які повинні параметрично залежати від числа обчислювальних елементів, що застосовуються для вирішення вихідної задачі.

### **49. У чому полягають основні дії на етапі розподілу підзадач по процесорах обчислювальної системи?**

Розподіл підзадач між обчислювальними елементами є завершальним етапом розробки паралельного методу. Слід зауважити, що управління розподілом навантаження для процесорів можливо тільки для обчислювальних систем з розподіленою пам’яттю, для мультипроцесорів розподіл навантаження зазвичай виконується операційною системою автоматично.

Очевидно, що розподіляти підзадачі потрібно так, щоб обсяг обчислень для кожного завантаження процесора був приблизно однаковим (балансування навантаження). Крім цього, розподіл підзадач повинно бути таким, щоб кількість комунікаційних взаємодій було мінімальним. Потрібно оцінити ефективність розроблюваного алгоритму. Для цієї мети слід визначити значення таких, наприклад, показників якості спроектованих паралельних обчислень, як прискорення обчислень, ступінь збалансованості завантаження процесорів, масштабованість наборів підзадач і т.д. ля цього загальну схемПісу складеного алгоритму піддають детальної програмної опрацюванні, щоб "розмістити" програми вирішення підзадач по процесорах відповідно до ви - лайливої схемою розпаралелювання. Далі програми запускають для виконання, при цьому виконання програми для кожного масиву підзадач називається процесом. Для реалізації комунікаційних взаємодій процеси повинні мати *канали передачі повідомлень* і *засобу обміну даними.*

### **50. У чому полягають основні методи передачі даних? Наведіть для цих методів аналітичні оцінки часу виконання?**

Найбільш прийняті методи передачі даних - за допомогою імпульсів: магнітних, електричних або світлових. Будь-який з цих методів має свої особливості та спрямований на конкретного споживача. Магнітні імпульси сьогодні найчастіше застосовуються для медичних досліджень. Вони фіксують та передають інформацію про внутрішніх органах. Це, наприклад – МРТ (магніторезонансний томограф) та інші види дослідження. Зараз вже канули у минуле бобіні та касетні магнітофони. Подекуди ще застосовуються комп'ютерні дискети (теж магнітний носій). Електричні імпульси використовується дуже широко в повсякденному житті. Це і телевізор, і монітор, і сигналізація та багато інших. Електричні імпульси можуть передавати аналоговий або цифровий сигнал, і навіть відразу кілька сигналів. Світлові імпульси, як спосіб передачі інформації, мають багаторазовий пріоритет у швидкості. У оптоволокні світловий імпульс може передавати дані зі швидкістю 1 Гбіт/с, практично без втрат. На даний момент величезну популярність набувають бездротові методи передачі даних. Виробляється вона за допомогою вузькосмугових та широкосмугових радіохвиль.

//аналітичні оцінки часу виконання не знаю

### **51. Які основні методи застосовуються при маршрутизації переданих даних по мережі?**

Є такі ***методи маршрутизації***: *статичні* та *динамічні*, *центра­лі­зовані,* *децентралізовані та змішані*.

При ***статичному*** методі маршрути розробляються і вводяться заздалегідь до програми роботи всіх вузлів мережі і не залежать від трафіку. При цьому оговорюються резервні варіанти, які включаються за розкладом або в екстрених випадках [81].

При ***динамічному*** управлінні маршрутизація здійснюється за допо­могою змінної таблиці маршрутизації. Маршрутизатор може сам визначати нові шляхи або модифікувати інформацію про старі. При цьому корекція даних про маршрути між станціями і збір інформації для прийняття рішень можуть бути проведені централізованими і децентралізованими методами [81].

При ***централізованому*** управлінні рішення про вибір маршруту приймається центром управління, наприклад сервером, та повідомляється усім вузлам, що знаходяться на даному маршруті.

При ***децентралізованому*** управлінні рішення про вибір маршруту кожним вузлом приймається автономно.

При ***змішаному*** управлінні рішення про вибір маршруту прий­мається вузлом з урахуванням рекомендацій центру управління.

### **52. У чому полягають основні способи досягнення паралелізму?**

Досягнення паралелізму можливе за умови виконання наступних вимог до архітектурних принципів побудови обчислювального середовища:

- незалежність функціонування окремих пристроїв ЕОМ - ця вимога відноситься всіх основних компонентів обчислювальної системи (пристрої введення-виведення, обробляючим процесорам, пристроям пам’яті);

- надлишковість елементів обчислювальної системи - організація надлишковості може здійснюватися у таких формах: використання спеціалізованих пристроїв (окремі процесори для цілочислової та дійсної арифметики, пристрої багаторівневої пам’яті - регістри, кеш); дублювання пристроїв ЕОМ шляхом використання декількох однотипних обробляючих процесорів чи кількох пристроїв оперативної пам’яті.

Окремою формою забезпечення паралелізму може бути конвеєрна реалізація обробляючих пристроїв, коли виконання операцій в пристроях представляється як виконання послідовності складових операції команд. В результаті при обчисленнях на таких пристроях на різних стадіях обробки можуть знаходитися одночасно декілька різних елементів даних.

### **53. У чому можуть полягати відмінності паралельних обчислювальних систем?**

**Паралельні обчислювальні системи** - це фізичні комп'ютерні, а також програмні системи, що реалізують той чи інший спосіб паралельну обробку даних на багатьох обчислювальних вузлах.

### Типи паралелізму

##### 1. Паралелізм на рівні бітів

Ця форма паралелізму заснована на збільшенні розміру машинного слова. Збільшення розміру машинного слова зменшує кількість операцій, необхідних процесору для виконання дій над змінними, чий розмір перевищує розмір машинного слова. До прикладу: на 8-бітному процесорі потрібно скласти два 16-бітових цілих числа. Для цього спочатку потрібно скласти нижні 8 біт чисел, потім скласти верхні 8 біт і до результату їх складання додати значення прапора переносу. Разом 3 інструкції. З 16-бітовим процесором можна виконати цю операцію однією інструкцією.

Історично 4-бітові мікропроцесори були замінені 8-бітними, потім з'явилися 16-бітові та 32-бітові. 32-бітові процесори довгий час були стандартом в повсякденних обчисленнях. З появою технології x86-64 для цих цілей стали використовувати 64-бітні процесори.

##### 2. Паралелізм на рівні інструкцій

Комп'ютерна програма - це, по суті, потік інструкцій, які виконуються процесором. Але можна змінити порядок цих інструкцій, розподілити їх за групами, які будуть виконуватися паралельно, без зміни результату роботи всієї програми. Даний прийом відомий як паралелізм на рівні інструкцій.

Сучасні процесори мають багатоступінчастий конвеєр команд. Кожній ступені конвеєра відповідає певна дія, виконувана процесором в цій інструкції на цьому етапі. Процесор з N ступенями конвеєра може мати одночасно до N різних інструкцій на різному рівні закінченості. Класичний приклад процесора з конвеєром - це RISC -процесор з 5-ма ступенями: вибірка інструкції з пам'яті (IF), декодування інструкції (ID), виконання інструкції (EX), доступ до пам'яті (MEM), запис результату в регістри (WB).

Деякі процесори, додатково до використання конвеєрів, володіють можливістю виконувати декілька інструкцій одночасно, що дає додатковий паралелізм на рівні інструкцій. Можлива реалізація даного методу за допомогою суперскалярної, коли інструкції можуть бути згруповані разом для паралельного виконання.

##### 3. Паралелізм даних

Основна ідея підходу, заснованого на паралелізмі даних, полягає в тому, що одна операція виконується відразу над всіма елементами масиву даних. Різні фрагменти такого масиву обробляються на векторному процесорі або на різних процесорах паралельної машини. Розподілом даних між процесорами займається програма. Векторизація або розпаралелювання в цьому випадку найчастіше виконується вже на етапі компіляції - перекладу вихідного тексту програми в машинні команди. Роль програміста в цьому випадку зазвичай зводиться до завдання налаштувань векторної або паралельної оптимізації компілятору, директив паралельної компіляції, використанню спеціалізованих мов для паралельних обчислень.

##### 4. Паралелізм завдань

Стиль програмування, заснований на паралелізмі завдань, має на увазі, що обчислювальна задача розбивається на декілька відносно самостійних підзадач і кожен процесор завантажується своєї власної підзадачі.

### **54. Що покладено в основу класифікації Флінна?**

В її основу покладено поняття потоку, під яким розуміється послідовність елементів, команд або даних, що обробляються процесором. Залежно від кількості потоків команд і потоків даних Флінн виділяє чотири класи архітектур:

1. SISD (Single Instruction Stream / Single Data Stream) - одиночний потік команд і одиночний потік даних.
2. MISD (Multiple Instruction Stream / Single Data Stream) - множинний потік команд і одиночний потік даних.
3. SIMD (Single Instruction Stream / Multiple Data Stream) - одиночний потік команд і множинний потік даних.
4. MIMD (Multiple Instruction Stream / Multiple Data Stream) - множинний потік команд і множинний потік даних.

### **55. У чому полягає принцип поділу багатопроцесорних систем на мультипроцесори і мультикомп’ютери?**

В залежності від організації підсистем оперативної пам'яті багатопроцесорні системи можна розділити на два наступних класи.

Системи із спільною (такою, що розділяється) пам'яттю — **мультипроцесори**, у яких є одна віртуальна пам'ять, а всі процесори мають однаковий доступ до даних та командам, що зберігаються в цій пам'яті . За цим принципом побудовані векторні паралельні процесори та симетричні мультипроцесори .   
  
Системи з розподіленою пам'яттю - **мультикомп'ютери**, у яких кожен процесор має свою локальну оперативну пам'ять, а в інших процесорів доступ до цієї пам'яті відсутній.

### **56. Які топології мереж передачі даних найбільш широко використовуються при побудові багатопроцесорних систем?**

1) Повний граф

2) Лінійка (Ланцюг)

3) Кільце

4) Зірка

5) Решітка (двовимірна чи тривимірна)

### **57. У чому полягають особливості мереж передачі даних для кластерів?**

Мережа кластера в першу чергу призначена не для зв'язку машин, а для зв'язку обчислювальних процесів. Тому чим вищою буде пропускна здатність мережі, тим швидше будуть вважатися паралельні завдання, запущені на кластері.

### **58. Які основні характеристики мереж передачі даних?**

найважливіші показники роботи мережі: продуктивність, надійність і безпека, розширюваність і масштабованість, прозорість, підтримка різних видів трафіку, характеристики якості обслуговування, керованість і сумісність.

Саме загальне побажання, яке можна висловити щодо роботи мережі - це виконання мережею того набору послуг, для надання яких вона призначена. Всі інші вимоги - продуктивність, надійність, сумісність, керованість, захищеність, розширюваність і масштабованість - пов'язані з якістю виконання цієї основної задачі.

Потенційно висока продуктивність - Це одна з основних переваг розподілених систем, до яких відносяться комп'ютерні мережі.

Основні характеристики продуктивності мережі:

В· час реакції;

В· швидкість передачі трафіку;

В· пропускна здатність;

В· затримка передачі і варіація затримки передачі.

У загальному випадку час реакції визначається як інтервал між виникненням запиту користувача до небудь мережевої служби і отриманням відповіді на нього.

Час реакції мережі зазвичай складається з декількох складових.

У загальному випадку в нього входить:

- час підготовки запитів на клієнтському комп'ютері;

- час передачі запитів між клієнтом і сервером через сегменти мережі і проміжне комунікаційне устаткування;

- час обробки запитів на сервері;

- час передачі відповідей від сервера клієнту і час обробки одержуваних від сервера відповідей на клієнтському комп'ютері.

Продуктивність мережі може характеризуватися також швидкістю передачі трафіку. Швидкість передачі трафіку може бути миттєвою, максимальною і середньою.

середня швидкість обчислюється шляхом розподілу загального обсягу переданих даних на час їх передач

Пропускна здатність - Максимально можлива швидкість обробки трафіку, певна стандартом технології, на якій побудована мережа. Пропускна здатність відображає максимально можливий обсяг даних, який передається мережею або її частиною в одиницю часу. На відміну від часу реакції або швидкості передачі трафіку пропускна здатність не залежить від завантаженості мережі і має постійне значення, обумовлене використовуваними в мережі технологіями.

Звичайно якість мережі характеризують величинами максимальної затримки передачі і варіацією затримки. Не всі типи трафіка чутливі до затримок передачі, у всякому разі, до тих величин затримок, які характерні для комп'ютерних мереж, - зазвичай затримки не перевищують сотень мілісекунд, рідше - кількох секунд.

# Практика Завдання

### Підготуйте огляд програмних бібліотек, що забезпечують виконання операцій передачі даних для систем з розподіленою пам'яттю.

**MPICH**:

Особливості:

* безкоштовна
* повна сумісність зі специфікацією MPI-1;
* наявність інтерфейсу в стилі MPI-2 з функціями для мови C++ зі специфікації MPI-1;
* часткова підтримка MPI-2;
* часткова підтримка паралельного вводу-виводу — ROMIO;
* підтримка великого числа архітектур, у тому числі кластерів, SMP і т. д.;
* наявність у складі MPICH тестів продуктивності і перевірки функціонування системи

Недоліки MPICH — неможливість запуску процесів під час роботи програми і відсутність засобів моніторингу поточного стану системи.

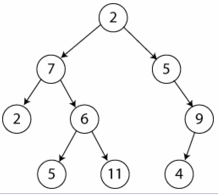
**Open MPI:**

В рамках проекту Open MPI визначає кілька цілей:

1. створити безкоштовне програмне забезпечення з відкритим вихідним кодом високої якості (повна реалізація MPI-3.0)
2. для того щоб забезпечити надзвичайно високу, конкурентоспроможну продуктивність (малий час затримки або з високою пропускною спроможністю)
3. залучати членів обчислювальних спільнот досягаючи при цьому розвитку за допомогою сторонніх осіб та отримуючи відгуки.
4. Надати стабільну платформу для сторонніх розробників та для комерційного використання.
5. допомогти запобігти "проблемі розгалуження", характерній для інших проектів MPI
6. для підтримки широкого спектру високопродуктивних обчислювальних платформ і середовищ

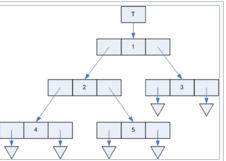
### 2. Розгляньте топологію мережі передачі даних у вигляді двійкового дерева.

**двійкове дерево** — структура даних у вигляді дерева, в якому кожна вершина має не більше двох дітей.

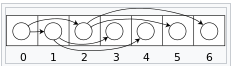


В залежності від задач, які вирішуються цими структурами та можливостей тої чи іншої мови програмування, існує декілька варіантів конструювання двійкових дерев.

Реалізація з використанням вказівників передбачає зберігання в кожній вершині дерева x, разом з даними двох полів (правим та лівим) right[x] та left[x], вказівників на відповідних дітей цієї вершини. Також іноді додається вказівник p[x] на батьківську вершину. Це спрощує деякі алгоритми та виявляється корисним, коли необхідно забезпечити швидкий доступ до батьківської вершини. Іноді достатньо тільки вказівника на батьківську вершину. Взагалі будь-яке орієнтоване дерево можна описати, знаючи тільки зв'язки від дітей до батьківської вершини.



Двійкові дерева також можуть бути побудовані на базі [масивів](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D0%B8%D0%B2_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)). Такий метод набагато ефективніший щодо економії пам'яті. В такому представленні, якщо вершина має порядковий номер i, то її діти знаходяться за індексами 2i+1 та 2i+2, а батьківська вершина за індексом ((i-1)/2) (за умов, що коренева вершина має індекс 0). Інший варіант зберігання дерева в масиві — зберігати індекси дітей.



### 3. Виділіть ефективно реалізовані класи задач для кожного типу топологій мережі передачі даних.

### 4. Розробіть модель і виконайте оцінку показників прискорення та ефективності паралельних обчислень:

#### для задачі скалярного добутку двох векторів,

#### b. для задачі пошуку максимального і мінімального значень для заданого набору числових даних,

#### c. для задачі знаходження середнього значення для заданого набору числових даних.

### 5. Виконайте відповідно до закону Амдаля оцінку максимально досяжного прискорення для задач п. 4.

### 6. Виконайте оцінку прискорення масштабування для задач п.4.

### 7. Виконайте побудову функцій ізоефективності для задач п. 4.

### 8. (\*) Розробіть модель і виконайте повний аналіз ефективності паралельних обчислень (прискорення, ефективність, максимально досяжний прискорення, прискорення масштабування, функція ізоефективності) для задачі множення матриці на вектор.

### 9. Розробіть алгоритми логічного представлення двійкового дерева для різних фізичних топологій мережі.

### 10. Розробіть алгоритми виконання основних операцій передачі даних для топології мережі у виді 3-мірної решітки.

### 11. Розробіть алгоритми виконання основних операцій передачі даних для топології мережі у виді двійкового дерева.

### 12. Розробіть програму-приклад для кожної наявної в MPI колективної операції.

### 13. Розробіть програму для знаходження мінімального (максимального) значення серед елементів вектора.

Припустимо, що на вхід програмі у командній строці першим елементом задано к-сть чисел у вектор, а далі послідовність елементів вектора. Алгоритм буде наступним:  
1) Кожен процес у комунікаторі, знаючи свій ранг та загалну к-сть процесів, визначає яку частину вхідного вектора він повинен опрацювати.

2) Послідовний перебір елементів відповідної частини масиву та визначення максимального та мінмального значення на цьому проміжку

3) Запис мінімального та максимального елемента у відповідний файл "FoundedMinMax\_index.txt"

4) Після опрацювання своєї частини масиву, нульовий процес повинен почекати до завершення роботи усіх процесів, а тоді опрацювати файли-результати обробки

5) Парсаючи кожен файл, він визначає мінімальний та максимальний серед них і виводить їх у фінальний файл “results.txt”.

Приблизна програма-реалізація на МРІ наведена нище

void main(int argc, char\* argv[])

{

MPI\_Init(&argc, &argv);

// Get the number of processes

int world\_size;

MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);

// Get the rank of the process

int world\_rank;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);

// omitting input-checking

long size = atoi(argv[1]);

long chunk = size / world\_size;

long begin = world\_rank \* chunk;

long end = world\_rank == world\_size - 1 ?

size - 1 : (world\_rank + 1) \* chunk - 1;

char filename[\_MAX\_FNAME];

sprintf\_s(filename, \_MAX\_FNAME, "FoundedMinMax\_%d.txt", world\_rank);

ofstream founded\_storage(filename);

long min = INT\_MAX;

long max = INT\_MIN;

for (unsigned i = begin; i < end; ++i)

{

long currNum = atoi(argv[i + 2]);

if (currNum < min)

min = currNum;

if (currNum > max)

max = currNum;

}

founded\_storage << "Min = " << min << endl;

founded\_storage << "Max = " << max << endl;

MPI\_Barrier(MPI\_COMM\_WORLD);

if (world\_rank == 0)

{

for (int i = 1; i < world\_size; ++i)

{

char filename[\_MAX\_FNAME];

sprintf\_s(filename, \_MAX\_FNAME, "FoundedMinMAx\_%d.txt", i);

ifstream read(filename);

long minFile = ParseMinFromFile(&read);

if (currNum < min)

min = currNum;

long maxFile = ParseMaxFromFile(&read);

if (currNum > max)

max = currNum;

}

ofstream write("results.txt");

write << "Min in vector = " << min << endl;

write << "Max in vector = " << max << endl;

}

MPI\_Finalize();

}

### 14. Розробіть програму для обчислення скалярного добутку двох векторів.

А тут не має бути паралелізм? має

int main()  
{  
 int i, n = 0;  
 double a = 0;  
 double b = 0;  
 double Smult = 0;  
 while(true)  
 {  
 cout<<"Enter size of vectors n = ";cin>>n;  
 for(Smult = (i = 0); i < n; i++)  
 {  
 cout<<"a["<<i + 1<<"] = ";cin>>a;  
 cout<<"b["<<i + 1<<"] = ";cin>>b;  
 Smult += a\*b;  
 }  
 cout<<"Scalar mult = "<<Smult<<endl;  
 }  
 return 0;  
}

### 15. Розробіть програму, в якій два процеси багаторазово обмінюються повідомленнями довжиною n байт. Виконайте експерименти та оцініть залежність часу виконання операції даних від довжини повідомлення. Порівняйте з теоретичними оцінками, побудованими за моделлю Хокні.

Процеси передають одне одному дані через сокети. Програма запускається вперше і в неї є дві вітки ходу. Через перевірку к-сті аргументів вхідного командного рядка спершу вона заходить в одну з віток і створює потік, котрий створює процес цієї ж програми, але з іншою к-стю аргументів, а потім створює сервер. Ця ж прогарма, отримавши іншу к-сть аргументів, створює вже не сервера, а клієнта. Потім сервер нескінченну к-сть раз передає клієнту таку к-сть байт, яка була введена користувачем з клавіатури при створенні сервера. (тільки замініть мій ip на свій).

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <stdlib.h>

#include <stdio.h>

#include <iostream>

// Need to link with Ws2\_32.lib, Mswsock.lib, and Advapi32.lib

#pragma comment (lib, "Ws2\_32.lib")

#pragma comment (lib, "Mswsock.lib")

#pragma comment (lib, "AdvApi32.lib")

#define DEFAULT\_PORT "27015"

int bufSize;

int ClientMode( int LEN)

{

WSADATA wsaData;

SOCKET ConnectSocket = INVALID\_SOCKET;

struct addrinfo \*result = NULL,

\*ptr = NULL,

hints;

char \*sendbuf = new char[LEN];

for (int i = 0; i < LEN -1; ++i)

{

sendbuf[i] = 'F';

}

sendbuf[LEN - 1] = '\0';

//std::cout << strlen(sendbuf);

char\* recvbuf = new char[LEN];

int iResult;

int recvbuflen = LEN;

// Validate the parameters

// Initialize Winsock

iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (iResult != 0) {

printf("WSAStartup failed with error: %d\n", iResult);

return 1;

}

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

// Resolve the server address and port

iResult = getaddrinfo("192.168.1.104", DEFAULT\_PORT, &hints, &result);

if (iResult != 0) {

printf("getaddrinfo failed with error: %d\n", iResult);

WSACleanup();

return 1;

}

// Attempt to connect to an address until one succeeds

for (ptr = result; ptr != NULL; ptr = ptr->ai\_next) {

// Create a SOCKET for connecting to server

ConnectSocket = socket(ptr->ai\_family, ptr->ai\_socktype,

ptr->ai\_protocol);

if (ConnectSocket == INVALID\_SOCKET) {

printf("socket failed with error: %ld\n", WSAGetLastError());

WSACleanup();

return 1;

}

// Connect to server.

iResult = connect(ConnectSocket, ptr->ai\_addr, (int)ptr->ai\_addrlen);

if (iResult == SOCKET\_ERROR) {

closesocket(ConnectSocket);

ConnectSocket = INVALID\_SOCKET;

continue;

}

break;

}

freeaddrinfo(result);

if (ConnectSocket == INVALID\_SOCKET) {

printf("Unable to connect to server!\n");

WSACleanup();

return 1;

}

// Send an initial buffer

iResult = send(ConnectSocket, sendbuf, (int)strlen(sendbuf), 0);

if (iResult == SOCKET\_ERROR) {

printf("send failed with error: %d\n", WSAGetLastError());

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

printf("Bytes Sent: %ld\n", iResult);

// shutdown the connection since no more data will be sent

iResult = shutdown(ConnectSocket, SD\_SEND);

if (iResult == SOCKET\_ERROR) {

printf("shutdown failed with error: %d\n", WSAGetLastError());

closesocket(ConnectSocket);

WSACleanup();

return 1;

}

// Receive until the peer closes the connection

do {

iResult = recv(ConnectSocket, recvbuf, recvbuflen, 0);

if (iResult > 0)

printf("Bytes received: %d\n", iResult);

else if (iResult == 0)

printf("Connection closed\n");

else

printf("recv failed with error: %d\n", WSAGetLastError());

} while (iResult > 0);

// cleanup

closesocket(ConnectSocket);

WSACleanup();

return 0;

}

int ServerMode()

{

WSADATA wsaData;

int iResult;

SOCKET ListenSocket = INVALID\_SOCKET;

SOCKET ClientSocket = INVALID\_SOCKET;

struct addrinfo \*result = NULL;

struct addrinfo hints;

int iSendResult;

char\* recvbuf = new char[bufSize];

int recvbuflen = bufSize;

for (int i = 0; i < bufSize; ++i)

{

recvbuf[i] = 'M';

recvbuf[bufSize - 1]= '\0';

}

// Initialize Winsock

iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (iResult != 0) {

printf("WSAStartup failed with error: %d\n", iResult);

return 1;

}

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

hints.ai\_flags = AI\_PASSIVE;

// Resolve the server address and port

iResult = getaddrinfo(NULL, DEFAULT\_PORT, &hints, &result);

if (iResult != 0) {

printf("getaddrinfo failed with error: %d\n", iResult);

WSACleanup();

return 1;

}

// Create a SOCKET for connecting to server

ListenSocket = socket(result->ai\_family, result->ai\_socktype, result->ai\_protocol);

if (ListenSocket == INVALID\_SOCKET) {

printf("socket failed with error: %ld\n", WSAGetLastError());

freeaddrinfo(result);

WSACleanup();

return 1;

}

// Setup the TCP listening socket

iResult = bind(ListenSocket, result->ai\_addr, (int)result->ai\_addrlen);

if (iResult == SOCKET\_ERROR) {

printf("bind failed with error: %d\n", WSAGetLastError());

freeaddrinfo(result);

closesocket(ListenSocket);

WSACleanup();

return 1;

}

freeaddrinfo(result);

iResult = listen(ListenSocket, SOMAXCONN);

if (iResult == SOCKET\_ERROR) {

printf("listen failed with error: %d\n", WSAGetLastError());

closesocket(ListenSocket);

WSACleanup();

return 1;

}

// Accept a client socket

ClientSocket = accept(ListenSocket, NULL, NULL);

if (ClientSocket == INVALID\_SOCKET) {

printf("accept failed with error: %d\n", WSAGetLastError());

closesocket(ListenSocket);

WSACleanup();

return 1;

}

// No longer need server socket

closesocket(ListenSocket);

// Receive until the peer shuts down the connection

iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);

if (iResult > 0) {

printf("Bytes received: %d\n", iResult);

}

do {

// Echo the buffer back to the sender

iSendResult = send(ClientSocket, recvbuf, recvbuflen, 0);

if (iSendResult == SOCKET\_ERROR) {

printf("send failed with error: %d\n", WSAGetLastError());

closesocket(ClientSocket);

WSACleanup(); system("pause");

return 1;

}

printf("Bytes sent: %d\n", iSendResult);

if (iResult == 0)

printf("Connection closing...\n");

} while (iResult > 0);

// shutdown the connection since we're done

iResult = shutdown(ClientSocket, SD\_SEND);

if (iResult == SOCKET\_ERROR) {

printf("shutdown failed with error: %d\n", WSAGetLastError());

closesocket(ClientSocket);

WSACleanup();

system("pause");

}

// cleanup

closesocket(ClientSocket);

WSACleanup();

system("pause");

return 0;

}

VOID WINAPI ForProcess(DWORD arg)

{

STARTUPINFO si;

PROCESS\_INFORMATION pi;

Sleep(bufSize\*5);

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);

ZeroMemory(&pi, sizeof(pi));

LPWSTR ptr = (LPWSTR)arg;

CreateProcess(NULL, // No module name (use command line)

ptr, // Command line

NULL, // Process handle not inheritable

NULL, // Thread handle not inheritable

FALSE, // Set handle inheritance to FALSE

0, // No creation flags

NULL, // Use parent's environment block

NULL, // Use parent's starting directory

&si, // Pointer to STARTUPINFO structure

&pi);

}

int \_\_cdecl main(int argc, char \*\*argv)

{

if (argc < 2) {

printf("usage: %s server-name\n", argv[0]);

return 1;

}

if (argc == 2)

{

std::cout << "Enter amount of bytes to keep sending it from server to client \n";

std::cin >> bufSize;

wchar\_t wtext[50];

swprintf(wtext, 40, L"Result.exe 2 %d", bufSize);

LPWSTR ptr = wtext;

CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ForProcess, ptr, 0, 0);

ServerMode();

return 0;

}

else

{

if (argc == 3)

ClientMode((int)argv[2]);

}

}

### 16. Розробіть програму, виконайте експерименти і порівняйте результати для різних алгоритмів реалізації операції збору, обробки та розсилки даних всіх процесам (функція MPI\_Allreduce).

Нехай у коммутаторі є набір процесів, кожен з яких має набір чисел одної вибірки. Потрібно порахувати девіанту цієї глобальної вибірки. Девіанта =sqrt ( сума квадратів від різниці кожного елемента вибірки і середнім значенням вибірки). Тому ми маємо обрахувати спершу середнє із усіх вибірок усіх процесів, а тоді суму усіх квадратів різниць. Далі наведений код із поясненнями:

rand\_nums = create\_rand\_nums(num\_elements\_per\_proc);  
  
// Sum the numbers locally  
float local\_sum = 0;  
int i;  
for (i = 0; i < num\_elements\_per\_proc; i++) {  
 local\_sum += rand\_nums[i];  
}  
  
// Reduce all of the local sums into the global sum in order to  
// calculate the mean  
float global\_sum;  
MPI\_Allreduce(&local\_sum, &global\_sum, 1, MPI\_FLOAT, MPI\_SUM,  
 MPI\_COMM\_WORLD);  
float mean = global\_sum / (num\_elements\_per\_proc \* world\_size);  
  
// Compute the local sum of the squared differences from the mean  
float local\_sq\_diff = 0;  
for (i = 0; i < num\_elements\_per\_proc; i++) {  
 local\_sq\_diff += (rand\_nums[i] - mean) \* (rand\_nums[i] - mean);  
}  
  
// Reduce the global sum of the squared differences to the root  
// process and print off the answer  
float global\_sq\_diff;  
MPI\_Reduce(&local\_sq\_diff, &global\_sq\_diff, 1, MPI\_FLOAT, MPI\_SUM, 0,  
 MPI\_COMM\_WORLD);  
  
// The standard deviation is the square root of the mean of the  
// squared differences.  
if (world\_rank == 0) {  
 float stddev = sqrt(global\_sq\_diff /  
 (num\_elements\_per\_proc \* world\_size));  
 printf("Mean - %f, Standard deviation = %f\n", mean, stddev);  
}

In the above code, each process computes the local\_sum of elements and sums them using MPI\_Allreduce. After the global sum is available on all processes, the mean is computed so that local\_sq\_diff can be computed. Once all of the local squared differences are computed, global\_sq\_diff is found by using MPI\_Reduce. The root process can then compute the standard deviation by taking the square root of the mean of the global squared differences.

### 17. Розробіть програму-приклад для кожного наявного в MPI способу конструювання похідних типів даних.

Способи конструювання типів в MPI

1. Неперервний – дає змогу визначити неперервний набір елементів існуючого типу як новий похідний тип.У разі неперервного способу використовується функція:

int **MPI\_Type\_contiguous**(int count,MPI\_Data\_type oldtype,MPI\_Datatype \*newtype).

Новий тип *newtype* створюється як *count* елементів вихідного типу *oldtype*. Наприклад, якщо вихідний тип даних має карту типу

{ (MPI\_INT,0),(MPI\_DOUBLE,8) },

то виклик функції *MPI\_Type\_contiguous* з параметрами

MPI\_Type\_contiguous (2, oldtype, &newtype);

Приведе до створення типу даних з картою типу

{ (MPI\_INT,0),(MPI\_DOUBLE,8),(MPI\_INT,16),(MPI\_DOUBLE,24) }.

Приклад

const int N =10;

double A[N][N];

double B[N][N];

MPI\_Datatype matrix;

MPI\_Type\_contiguous(N\*N, MPI\_DOUBLE, &matrix);

MPI\_Type\_commit(&matrix);

if(rank == master\_rank

MPI\_Send(A,1, matrix,1,10, comm);

else if( rank ==1)

MPI\_Recv(B,1, matrix,0,10, comm, &status);

2. Векторний – забезпечує створення нового похідного типу як набору елементів існуючого типу, між елементами якого є регулярні проміжки пам’яті.

Використовується функція

int **MPI\_Type\_vector** ( int count, int blocklen, int stride, MPI\_Data\_type oldtype, MPI\_Datatype \*newtype )

де

- **count** – кількість блоків,

- **blocklen** –розмір кожного блока,

- **stride** –кількість елементів,розташованих між двома сусідніми блоками

- **oldtype** -вихідний тип,

- **newtype** -новий тип.

int **MPI\_Type\_hvector** ( int count, int blocklen, MPI\_Aint stride, MPI\_Data\_type oldtype, MPI\_Datatype \*newtype ).

Відрізняється тим, що параметр *stride* для визначення інтервалу між блоками задається в байтах, а не в елементах вихідного типу.

Приклади:

• Конструювання типу для виділення половини (тільки парних або непарних) рядків матриці *nxn*:

MPI\_Type\_vector ( n/2, n, 2\*n, &StripRowType, &ElemType ),

• Конструювання типу для виділення стовпця матриці *nxn*:

MPI\_Type\_vector ( n, 1, n, &ColumnType, &ElemType ),

• Конструювання типу для виділення головної діагоналі матриці *nxn*:

MPI\_Type\_vector ( n, 1, n+1, &DiagonalType, &ElemType ).

3. Індексний – відрізняється від векторного тим, що проміжки між елементами вихідного типу можуть мати нерегулярний характер.

Використовується функції

int **MPI\_Type\_indexed** ( int count, int blocklens[], int indices[], MPI\_Data\_type oldtype, MPI\_Datatype \*newtype ),

де

- **count**– кількість блоків,

- **blocklens** –кількість елементів в кожному блоці,

- **indices** –зміщення кожного блоку від початку типу,

- **oldtype** -вихідний тип,

- **newtype** -новий тип.

int **MPI\_Type\_hindexed** ( int count, int blocklens[], MPI\_Aint indices[], MPI\_Data\_type oldtype, MPI\_Datatype \*newtype )

(інтервали між блоками задаються в байтах)

Приклад - конструювання типу для опису верхньої трикутної матриці n\*n

for ( i=0, i<n; i++ ) {

blocklens[i] = n - i;

indices[i] = i \* n + i;

}

MPI\_Type\_indexed ( n, blocklens, indices, &UTMatrixType, &ElгemType ).

4. Структурний – забезпечує найбільш загальний опис похідного типу через явне вказання карти створюваного типу даних.

int MPI\_Type\_struct ( int count, int blocklens[], MPI\_Aint indices[], MPI\_Data\_type oldtypes[], MPI\_Datatype \*newtype ),

где

- count– кількість блоків,

- blocklens – кількість елементів в кожному блоці,

- indices– зміщення кожного блоку від початку типу (в байтах),

- oldtypes - вихідні типи,

- newtype- новий типе.

Приклад

blen[0] = 1;

indices[0] = 0;

oldtypes[0] = MPI\_INT;

blen[1] = 1;

indices[1] = &data.b — &data;

oldtypes[1] = MPI\_CHAR;

blen[2] = 1;

indices[2] = sizeof(data);

oldtypes[2] = MPI\_FLOAT;

MPI\_Type\_struct(3, blen, indices, oldtypes, &newtype);

### 18. Розробіть програму-приклад з використанням функцій упаковки і розпаковування даних. Виконайте експерименти і порівняйте з результатами при використанні похідних типів даних.

### 19. Розробіть похідні типи даних для рядків, стовпців, діагоналей матриць.

//не впевнена, зроблено за аналогією з однієї книжки

• Конструювання типу для виділення рядка матриці *nxn*:

MPI\_Type\_vector ( n, n, n, &RowType, &ElemType ),

• Конструювання типу для виділення стовпця матриці *nxn*:

MPI\_Type\_vector ( n, 1, n, &ColumnType, &ElemType ),

• Конструювання типу для виділення головної діагоналі матриці *nxn*:

MPI\_Type\_vector ( n, 1, n+1, &DiagonalType, &ElemType ).

### **20. Розробіть програму-приклад для кожної з розглянутих функцій для управління процесами і комунікаторами.**

На цьому сайті круто пояснено: <http://www.intuit.ru/studies/courses/4447/983/lecture/14927?page=7>  
  
Хто буде це питання робити - можете туди дивитись. Нище копійнув головні частини із сайту, але там в більшості опис функцій, але без прикладів

Группы процессов могут быть созданы только из уже существующих групп. В качестве исходной группы может быть использована группа, связанная с предопределенным коммуникатором MPI\_COMM\_WORLD.

Для получения группы, связанной с существующим коммуникатором, используется функция:

int MPI\_Comm\_group( MPI\_Comm comm, MPI\_Group \*group ).

Далее, на основе существующих групп, могут быть созданы новые группы:

* создание новой группы **newgroup** из существующей группы **oldgroup**, которая будет включать в себя n процессов, ранги которых перечисляются в массиве **ranks**:
* int MPI\_Group\_incl(MPI\_Group oldgroup,int n, int \*ranks,MPI\_Group \*newgroup)
* ,
* создание новой группы **newgroup** из группы **oldgroup**, которая будет включать в себя n процессов, ранги которых не совпадают с рангами, перечисленными в массиве **ranks**:
* int MPI\_Group\_excl(MPI\_Group oldgroup,int n, int \*ranks,MPI\_Group \*newgroup).

Для получения новых групп над имеющимися группами процессов могут быть выполнены операции объединения, пересечения и разности:

* создание новой группы **newgroup** как объединения групп **group1** и **group2**:
* int MPI\_Group\_union(MPI\_Group group1, MPI\_Group group2, MPI\_Group \*newgroup)
* ;
* создание новой группы **newgroup** как пересечения групп **group1** и **group2**:
* int MPI\_Group\_intersection ( MPI\_Group group1, MPI\_Group group2,  
  MPI\_Group \*newgroup ),
* создание новой группы **newgroup** как разности групп **group1** и **group2**:
* int MPI\_Group\_difference ( MPI\_Group group1, MPI\_Group group2,  
  MPI\_Group \*newgroup ).

При конструировании групп может оказаться полезной специальная пустая группа MPI\_COMM\_EMPTY.

Ряд функций MPI обеспечивает получение информации о группе процессов:

* получение количества процессов в группе:
* int MPI\_Group\_size ( MPI\_Group group, int \*size ),
* получение ранга текущего процесса в группе:
* int MPI\_Group\_rank ( MPI\_Group group, int \*rank ).

После завершения использования группа должна быть удалена:

int MPI\_Group\_free ( MPI\_Group \*group )

(выполнение данной операции не затрагивает коммуникаторы, в которых используется удаляемая группа).

==================================================================

Для пояснения рассмотренных функций можно привести пример создания коммуникатора, в котором содержатся все процессы, кроме процесса, имеющего ранг 0 в коммуникаторе MPI\_COMM\_WORLD (такой коммуникатор может быть полезен для поддержки схемы организации параллельных вычислений "менеджер - исполнители" – см. раздел 6):

MPI\_Group WorldGroup, WorkerGroup;  
MPI\_Comm Workers;  
int ranks[1];  
ranks[0] = 0;  
// получение группы процессов в MPI\_COMM\_WORLD  
MPI\_Comm\_group(MPI\_COMM\_WORLD, &WorldGroup);  
// создание группы без процесса с рангом 0  
MPI\_Group\_excl(WorldGroup, 1, ranks, &WorkerGroup);  
// Создание коммуникатора по группе  
MPI\_Comm\_create(MPI\_COMM\_WORLD,WorkerGroup,&Workers);  
...  
MPI\_Group\_free(&WorkerGroup);  
MPI\_Comm\_free(&Workers);

В качестве примера можно рассмотреть задачу представления набора процессов в виде двумерной решетки. Пусть **p=q\*q** есть общее количество процессов, следующий далее фрагмент программы обеспечивает получение коммуникаторов для каждой строки создаваемой топологии:

MPI\_Comm comm;  
int rank, row;  
MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);  
row = rank/q;  
MPI\_Comm\_split(MPI\_COMM\_WORLD,row,rank,&comm);

При выполнении данного примера, например, при **p=9**, процессы с рангами (0,1,2) образуют первый коммуникатор, процессы с рангами (3,4,5) – второй и т.д.

### **21. Розробіть програму для представлення безлічі процесів у вигляді прямокутної решітки. Створіть комунікатори для кожного рядка і стовпця процесів. Виконайте колективну операцію для всіх процесів і для одного з створених комунікаторів. Порівняйте час виконання операції.**

Хто буде робити це завдання, тут класна інфа. Поки скидую нефільтроване

<http://www.intuit.ru/studies/courses/4447/983/lecture/14927?page=7>

**Декартовы топологии**, в которых множество процессов представляется в виде прямоугольной **решетки** (см. п. 1.4.1 и [рис. 5.7](http://www.intuit.ru/studies/courses/4447/983/lecture/14927?page=5#image.5.7)), а для указания процессов используется декартова система координат, широко применяются во многих задачах для описания структуры имеющихся информационных зависимостей. В числе примеров таких задач – матричные алгоритмы (см. разделы 7 и 8) и сеточные методы решения дифференциальных уравнений в частных производных (см. раздел 12).

Для создания декартовой топологии (решетки) в MPI предназначена функция:

int MPI\_Cart\_create(MPI\_Comm oldcomm, int ndims, int \*dims, int \*periods,  
int reorder, MPI\_Comm \*cartcomm),

где: oldcomm - исходный коммуникатор, ndims - размерность декартовой решетки, dims - массив длины ndims, задает количество процессов в каждом измерении решетки, periods - массив длины ndims, определяет, является ли решетка периодической вдоль каждого измерения, reorder - параметр допустимости изменения нумерации процессов, cartcomm – создаваемый коммуникатор с декартовой топологией процессов.

Операция создания топологии является коллективной и, тем самым, должна выполняться всеми процессами исходного коммуникатора.

Для пояснения назначения параметров функции **MPI\_Cart\_create** рассмотрим пример создания двухмерной решетки **4x4**, в которой строки и столбцы имеют кольцевую структуру (за последним процессом следует первый процесс):

// создание двухмерной решетки 4x4  
MPI\_Comm GridComm;  
int dims[2], periods[2], reorder = 1;  
dims[0] = dims[1] = 4;  
periods[0] = periods[1] = 1;  
MPI\_Cart\_create(MPI\_COMM\_WORLD, 2, dims, periods, reoreder, &GridComm);

Следует отметить, что в силу кольцевой структуры измерений сформированная в рамках примера топология является **тором**.

Для определения декартовых координат процесса по его рангу можно воспользоваться функцией:

int MPI\_Card\_coords(MPI\_Comm comm,int rank,int ndims,int \*coords),

где: comm – коммуникатор с топологией решетки, rank - ранг процесса, для которого определяются декартовы координаты, ndims - размерность решетки, coords - возвращаемые функцией декартовы координаты процесса.

Обратное действие – определение ранга процесса по его декартовым координатам – обеспечивается при помощи функции:

int MPI\_Cart\_rank(MPI\_Comm comm, int \*coords, int \*rank),

где comm – коммуникатор с топологией решетки, coords - декартовы координаты процесса, rank - возвращаемый функцией ранг процесса.

Полезная во многих приложениях процедура разбиения решетки на подрешетки меньшей размерности обеспечивается при помощи функции:

int MPI\_Card\_sub(MPI\_Comm comm, int \*subdims, MPI\_Comm \*newcomm),

где: comm - исходный коммуникатор с топологией решетки, subdims – массив для указания, какие измерения должны остаться в создаваемой подрешетке, newcomm - создаваемый коммуникатор с подрешеткой.

Операция создания подрешеток также является коллективной и, тем самым, должна выполняться всеми процессами исходного коммуникатора. В ходе своего выполнения функция **MPI\_Cart\_sub** определяет коммуникаторы для каждого сочетания координат фиксированных измерений исходной решетки.

Для пояснения функции **MPI\_Cart\_sub** дополним ранее рассмотренный пример создания двухмерной решетки и определим коммуникаторы с декартовой топологией для каждой строки и столбца решетки в отдельности:

// создание коммуникаторов для каждой строки и столбца решетки  
MPI\_Comm RowComm, ColComm;  
int subdims[2];  
// создание коммуникаторов для строк  
subdims[0] = 0; // фиксации измерения  
subdims[1] = 1; // наличие данного измерения в подрешетке  
MPI\_Cart\_sub(GridComm, subdims, &RowComm);  
// создание коммуникаторов для столбцов  
subdims[0] = 1;  
subdims[1] = 0;  
MPI\_Cart\_sub(GridComm, subdims, &ColComm);

В приведенном примере для решетки размером 4х4 создаются 8 коммуникаторов, по одному для каждой строки и столбца решетки. Для каждого процесса определяемые коммуникаторы **RowComm** и **ColComm** соответствуют строке и столбцу процессов, к которым данный процесс принадлежит.

Дополнительная функция **MPI\_Cart\_shift** обеспечивает поддержку процедуры последовательной передачи данных по одному из измерений решетки (**операция сдвига данных** - см. раздел 3). В зависимости от периодичности измерения решетки, по которому выполняется сдвиг, различаются два типа данной операции:

* **Циклический сдвиг** на k элементов вдоль измерения решетки – в этой операции данные от процесса **i** пересылаются процессу **(i+k) mod dim**, где **dim** есть размер измерения, вдоль которого производится сдвиг,
* **Линейный сдвиг** на k позиций вдоль измерения решетки – в этом варианте операции данные от процессора **i** пересылаются процессору **i+k** (если таковой существует).

Функция **MPI\_Cart\_shift** обеспечивает получение рангов процессов, с которыми текущий процесс (процесс, вызвавший функцию **MPI\_Cart\_shift**) должен выполнить обмен данными:

int MPI\_Card\_shift(MPI\_Comm comm, int dir, int disp,  
int \*source, int \*dst),

где: comm – коммуникатор с топологией решетки, dir - номер измерения, по которому выполняется сдвиг, disp - величина сдвига (<0 – сдвиг к началу измерения), source – ранг процесса, от которого должны быть получены данные, dst - ранг процесса которому должны быть отправлены данные.

Следует отметить, что функция **MPI\_Cart\_shift** только определяет ранги процессов, между которыми должен быть выполнен обмен данными в ходе операции сдвига. Непосредственная передачами данных, может быть выполнена, например, при помощи функции MPI\_Sendrecv.

### **22. Розробіть програми-приклади для передачі даних між процесами різних комунікаторів.**

**Приклад 1**: `` Конвеєр '' з трьох груп  
  
Обмінюються групи 0 і 1, потім групи 1 і 2. Тому, група 0 вимагає один інтер-комунікатор, група 1 вимагає два інтер-комунікатора і група 2 вимагає 1 інтер-комунікатор.

main(int argc, char \*\*argv)  
 {  
 MPI\_Comm myComm; /\* интракоммуникатор локальной  
 подгруппы \*/  
 MPI\_Comm myFirstComm; /\* интеркоммуникатор \*/  
 MPI\_Comm mySecondComm; /\* второй интеркоммуникатор  
 (только группа 1) \*/  
 int membershipKey;  
 int rank;  
  
 MPI\_Init(&argc, &argv);  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  
  
 /\* код пользователя обязан генерировать ключ  
 принадлежности в диапазоне [0, 1, 2] \*/  
 membershipKey = rank % 3;  
  
 /\* формирование интра-коммуникатора для локальной подгруппы \*/  
  
 MPI\_Comm\_split(MPI\_COMM\_WORLD, membershipKey, rank, &myComm);  
  
 /\* формирование интер-коммуникаторов  
 \*/  
  
 if (membershipKey == 0)  
 { /\* Группа 0 связывается с группой 1 \*/  
  
 MPI\_Intercomm\_create(myComm, 0,  
 MPI\_COMM\_WORLD, 1, 1, &myFirstComm);  
 }  
 else if (membershipKey == 1)  
 { /\* Группа 1 связывается с группами 0 и 2. \*/   
  
 MPI\_Intercomm\_create(myComm, 0, MPI\_COMM\_WORLD,  
 0, 1, &myFirstComm);   
 MPI\_Intercomm\_create(myComm, 0, MPI\_COMM\_WORLD,  
 2, 12, &mySecondComm);   
 }   
 else if (membershipKey == 2)   
 { /\* Группа 2 связывается с группой 1. \*/   
  
 MPI\_Intercomm\_create(myComm, 0, MPI\_COMM\_WORLD,  
 1, 12, &myFirstComm);   
 }   
   
 /\* рабочий участок ... \*/   
   
 switch(membershipKey) /\* удаление коммуникаторов  
 \*/   
 {   
 case 1:   
 MPI\_Comm\_free(&mySecondComm);   
 case 0:   
 case 2:   
 MPI\_Comm\_free(&myFirstComm);   
 break;   
 }   
   
 MPI\_Finalize();   
 }  
**Приклад 2:** Кільце з трьох груп  
  
Обмінюються групи 0 і 1, групи 1 і 2, групи 0 і 2. Отже, кожна група вимагає два інтеркоммунікатора.

main(int argc, char \*\*argv)  
 {  
 MPI\_Comm myComm; /\* интра-коммуникатор для локальной   
 подгруппы \*/   
 MPI\_Comm myFirstComm; /\* интер-коммуникатор \*/   
 MPI\_Comm mySecondComm;   
 MPI\_Status status;   
 int membershipKey;   
 int rank;   
   
 MPI\_Init(&argc, &argv);   
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);   
 ...   
   
 /\* код пользователя должен генерировать ключ принадлежности в  
 диапазоне [0, 1, 2] \*/   
  
 membershipKey = rank % 3;   
   
 /\* формирование интра-коммуникатора для локальной подгруппы \*/   
 MPI\_Comm\_split(MPI\_COMM\_WORLD, membershipKey, rank, &myComm);   
   
 /\* формирование интер-коммуникаторов. \*/   
 if (membershipKey == 0)   
 { /\* Группа 0 связывается с группами 1 и 2. \*/   
 MPI\_Intercomm\_create(myComm, 0, MPI\_COMM\_WORLD, 1,   
 1, &myFirstComm);   
 MPI\_Intercomm\_create(myComm, 0, MPI\_COMM\_WORLD, 2,   
 2, &mySecondComm);   
 }   
 else if (membershipKey == 1)   
 { /\* Группа 1 связывается с группами 0 и 2. \*/   
 MPI\_Intercomm\_create(myComm, 0, MPI\_COMM\_WORLD, 0,   
 1, &myFirstComm);   
 MPI\_Intercomm\_create(myComm, 0, MPI\_COMM\_WORLD, 2,   
 12, &mySecondComm);   
 }   
 else if (membershipKey == 2)   
 { /\* Группа 1 связывается с группами 0 и 1. \*/   
 MPI\_Intercomm\_create(myComm, 0, MPI\_COMM\_WORLD, 0,   
 2, &myFirstComm);  
 MPI\_Intercomm\_create(myComm, 0, MPI\_COMM\_WORLD, 1,   
 12, &mySecondComm);   
 }   
   
 /\* выполнение некоторой работы ... \*/   
   
 /\* Теперь освобождаем коммуникаторы перед окончанием... \*/   
 MPI\_Comm\_free(&myFirstComm);   
 MPI\_Comm\_free(&mySecondComm);   
 MPI\_Comm\_free(&myComm);   
 MPI\_Finalize();   
 }

### 23. Розробіть програму-приклад для декартової топології.

Для создания декартовой топологии (решетки) в MPI предназначена функция:

int MPI\_Cart\_create(MPI\_Comm oldcomm, int ndims, int \*dims,   
 int \*periods, int reorder, MPI\_Comm \*cartcomm),

где

* **oldcomm** — исходный коммуникатор;
* **ndims** — размерность декартовой решетки;
* **dims** — массив длины ndims, задает количество процессов в каждом измерении решетки;
* **periods** — массив длины ndims, определяет, является ли решетка периодической вдоль каждого измерения;
* **reorder** — параметр допустимости изменения нумерации процессов ;
* **cartcomm** — создаваемый коммуникатор с декартовой топологией процессов.

Для пояснения назначения параметров функции MPI\_Cart\_create рассмотрим пример создания двумерной решетки 4x4, в которой строки и столбцы имеют кольцевую структуру (за последним процессом следует первый процесс ):

// Создание двумерной решетки 4x4  
MPI\_Comm GridComm;  
int dims[2], periods[2], reorder = 1;  
dims[0] = dims[1] = 4;  
periods[0] = periods[1] = 1;  
MPI\_Cart\_create(MPI\_COMM\_WORLD, 2, dims, periods, reorder,  
 &GridComm);

### **24. Розробіть програму-приклад для топології графа.**

Час виконання для звичайного алгоритму топологічного сортування лінійний до кількості вершин плюс кількість ребер {O(|V|+|E|).}

Один з цих алгоритмів працює, вибираючи вершини в тому самому порядку, що і випадкове топологічне сортування. Спочатку знаходить набір «початкових вершин», які не мають ребер, що входять, і вставляє їх в набір S; щонайменше одна така вершина має існувати, якщо граф ациклічний. Тоді:

L ← Порожній список, що буде містити відсортовані елементи  
S ← Набір вершин без ребер, що входять  
**доки** S не порожнє **виконати**  
 видалити вершину n з S  
 вставити n в L  
 **для кожної** вершини m з ребром *e* з n до m **виконувати**  
 видалити ребро e з графа  
 **якщо** m не має більше ребер, що входять **тоді**  
 вставити m в S  
**якщо** граф має ребра **тоді**  
 вивести повідомлення про помилку (у графа є як мінімум один цикл)  
**інакше**   
 вивести повідомлення (пропоноване топологічне сортування: L)

Якщо маємо справу з [орієнтованим ациклічним графом](https://uk.wikipedia.org/wiki/%D0%9E%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B9_%D0%B0%D1%86%D0%B8%D0%BA%D0%BB%D1%96%D1%87%D0%BD%D0%B8%D0%B9_%D0%B3%D1%80%D0%B0%D1%84), то алгоритм видасть рішення (не унікальне).

Альтернативний алгоритм базується на [пошуку в глибину](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%88%D1%83%D0%BA_%D0%B2_%D0%B3%D0%BB%D0%B8%D0%B1%D0%B8%D0%BD%D1%83). Для цього алгоритму ребра вказуються у зворотному напрямку. Тобто якщо ребро іде з x до y, то це означає, що робота x залежить від роботи y (іншими словами робота y має бути завершена перед тим, як x зможе стартувати). Алгоритм проходить кожну вершину в графі в довільному порядку, започатковуючи пошук у глибину, що закінчується коли досягає вершину, яку вже відвідали з початку сортування:

L ← Порожній список, що буде містити відсортований набір вершин  
S ← Набір всіх вершин

**функція** відвідати(вершина n)  
 **якщо** n ще не була відвідана **тоді**  
 помітити n як відвідану  
 **для кожної** вершини m з ребром від n до m **виконати**  
 відвідати(m)  
 додати n до L

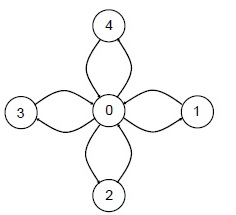
**для кожної** вершини n в S **виконати**  
 відвідати(n)

### **25. Розробіть підпрограми для створення деякого набору додаткових віртуальних топологій (зірка, дерево та ін.).**

### **Для создания коммуникатора с топологией типа граф в MPI предназначена функция**

### int MPI\_Graph\_create(MPI\_Comm oldcomm, int nnodes, int \*index, int \*edges,int reorder, MPI\_Comm \*graphcomm),

где: oldcomm - исходный коммуникатор, nnodes - количество вершин графа, index - количество исходящих дуг для каждой вершины,edges - последовательный список дуг графа, reorder - параметр допустимости изменения нумерации процессов, cartcomm – создаваемый коммуникатор с топологией типа граф.



Для примера создадим топологию графа со структурой ЗІРКА, представленной на рис. В этом случае количество процессов равно 5, порядки вершин (количества исходящих дуг) принимают значения (4,1,1,1,1), а матрица инцидентности (номера вершин, для которых дуги являются входящими) имеет вид:

|  |  |
| --- | --- |
| **Процессы** | **Линии связи** |
| 0 | 1, 2, 3, 4 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |

Для создания топологии с графом данного вида необходимо выполнить следующий программный код:

// создание топологии типа звезда  
int index[] = { 4,1,1,1,1 };  
int edges[] = { 1,2,3,4,0,0,0,0 };  
MPI\_Comm StarComm;  
MPI\_Graph\_create(MPI\_COMM\_WORLD, 5, index, edges, 1, &StarComm);

Приведем еще две полезные функции для работы с топологиями графа. Количество соседних процессов, в которых от проверяемого процесса есть выходящие дуги, может быть получено при помощи функции:

int MPI\_Graph\_neighbors\_count(MPI\_Comm comm,int rank, int \*nneighbors).

Получение рангов соседних вершин обеспечивается функцией:

int MPI\_Graph\_neighbors(MPI\_Comm comm,int rank,int mneighbors, int \*neighbors),

где **mneighbors** есть размер массива **neighbors**.

Дерево - це вироджений граф (тобто, частковий випадок), тому використовуючи функції вище - можна забацати граф, який буде деревом))

### 26. Виконайте реалізацію двох стрічкових алгоритмів множення матриць. Порівняйте часи виконання цих алгоритмів.

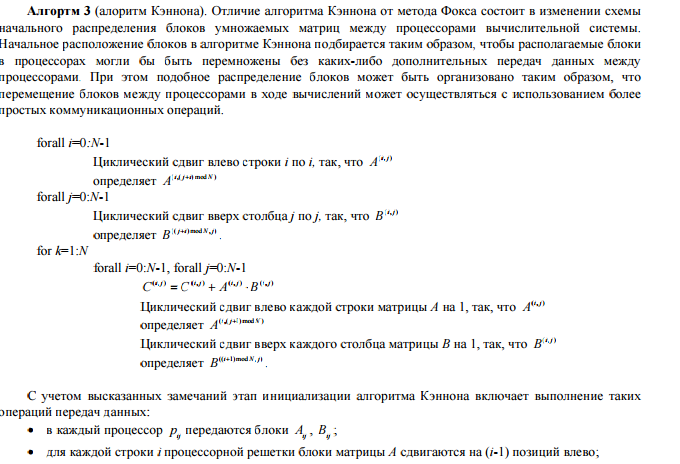
### Для обчислення одного рядка матриці С необхідно, щоб в кожній підзадачі містився рядок матриці А і був забезпечений доступ до усіх стовпців матриці B. Можливі способи організації паралельних обчислень полягають в наступному. 1) Перший алгоритм. Алгоритм є ітераційною процедурою, кількість ітерацій якої співпадає з числом підзадач. На кожній ітерації алгоритму кожна підзадача містить по одному рядку матриці А і одному стовпцю матриці В. При виконанні ітерації проводиться скалярне множення рядків, що містяться в підзадачах, і стовпців, що призводить до отримання відповідних елементів результуючої матриці С. Після закінчення обчислень у кінці кожної ітерації стовпці матриці В мають бути передані між підзадачами так, щоб у кожній підзадачі опинилися нові стовпці матриці В і могли бути вичислені нові елементи матриці C. При цьому така передача стовпців між підзадачами має бути організована так, щоб після завершення ітерацій алгоритму в кожній підзадачі послідовно опинилися усі стовпці матриці В.

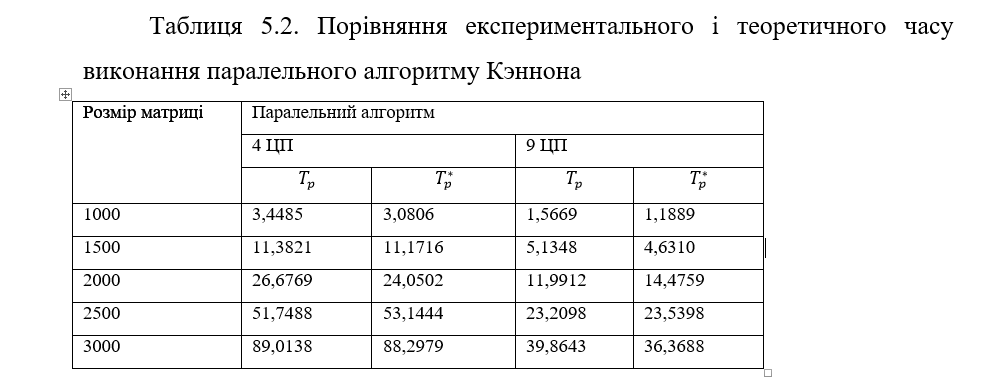
2) Другий алгоритм. Відмінність другого алгоритму полягає в тому, що в підзадачах розташовуються не стовпці, а рядки матриці B. Як результат, перемножування даних кожної підзадачі зводиться не до скалярного множення наявних векторів, а до їх по елементного множення. В результаті подібного множення в кожній підзадачі виходить рядок часткових результатів для матриці C.

Відповідно до отриманих співвідношень загальний час виконання паралельного алгоритму матричного множення визначається наступним вираженням:

T=(n2/p)\*(2n-1)\*t+(α+w\*n\*(n/p)/b)  
де α – латентність, а w – розмір елементу матриці в байтах,p-кількість процесорів

### 27. Виконайте реалізацію алгоритму Кеннона. Побудуйте теоретичні оцінки часу роботи цього алгоритму з урахуванням параметрів використовуваної обчислювальної системи. Проведіть обчислювальні експерименти. Порівняйте результати реальних експериментів з раніше отриманими теоретичними оцінками.





### **28. Виконайте реалізацію блокових алгоритмів множення матриць, які могли б бути виконані для прямокутних процесорних решіток загального вигляду.**

### **29. Виконайте реалізацію матричного множення з використанням раніше розроблених програм множення матриці на вектор.**

### **30. Виконайте аналіз ефективності паралельних обчислень окремо для прямого і зворотного етапів методу Гауса. Оцініть, на якому етапі відбувається більше зниження показників.**

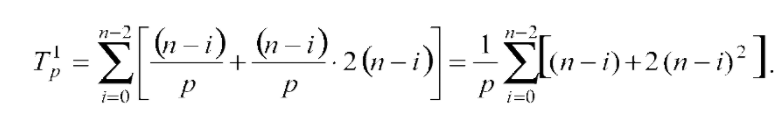
ПРЯМИЙ ХІД

Час виконання послідовного варіанту прямого ходу методу Гауса складає:

=(2\*(n^3))/3.

Визначимо складність паралельного варіанту прямого ходу методу Гауса

На кожній ітерації для вибору провідного рядка кожен процесор повинен зробити вибір максимального значення в стовпці з невідомою, якої ми позбуваємося, в межах своєї смуги. Поточний розмір смуги приблизно можна оцінити як (n-i)/p, де i, 0≤ i<n-1, номер ітерації промяго ходу Гауса. Далі кожен процесор повинен виконати віднімання провідного рядка від кожного рядка з тих що залишилися в смузі цього процесора.

Складність процедури віднімання рядків можна оцінити як 2\*(n-i) операцій. З урахуванням загального числа ітерацій кількість операцій паралельного варіанту прямого ходу Гауса становить

Показник прискорення і ефективності:

=

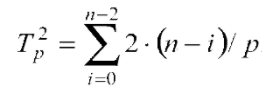
E/p

ОБЕРНЕНИЙ ХІД

Час виконання послідовного варіанту оберненого ходу методу Гауса складає:

= n^2

На кожній ітерації оберненого ходу алгоритму Гауса після розсилки вирахуваного значення чергової невідомої кожен процесор повинен оновити значення правих частин для всіх рядків розміщених в цьому процесорі. Звідси випливає, що алгоритм оберненого ходе можна оцінити як



Показник прискорення і ефективності:

=

E/p

### **31. Виконайте розробку паралельного варіанта методу Гауса при вертикальному розбитті матриці по стовпцях. Побудуйте теоретичні оцінки часу роботи цього алгоритму з урахуванням параметрів використовуваної обчислювальної системи. Проведіть обчислювальні експерименти. Порівняйте результати виконаних експериментів з раніше отриманими теоретичними оцінками.**

### 32. Виконайте реалізацію паралельного методу сполучених градієнтів. Побудуйте теоретичні оцінки часу роботи цього алгоритму з урахуванням параметрів використовуваної обчислювальної системи. Проведіть обчислювальні експерименти. Порівняйте результати виконаних експериментів з раніше отриманими теоретичними оцінками.

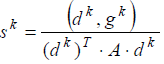
1. обчислення градієнта



2. Обчислення вектора напрямку

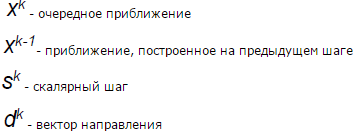


3. Обчислення величини зсуву по заданому напрямку



4. Обчислення нового наближення

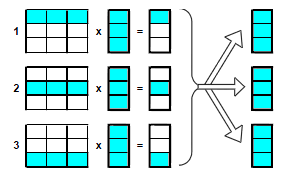




Виконання ітерацій методу здійснюється послідовно, отже найбільш доцільний підхід полягає в розпаралелювання обчислень, що реалізуються в ході виконання окремих ітерацій. Основні обчислення, що виконуються відповідно до методу, складаються в перемножении матриці А на вектора x і d. Додаткові обчислення, мають менший порядок складності є різні операції обробки векторів (скалярний твір, додавання і віднімання, множення на скаляр).

У послідовному алгоритмі всього 2 множення матриці на вектор на кроках 1 і 3, саме тут і задіюється паралельна схема. Розглянемо паралельний варіант множення матриці на вектор. Розподіл даних-розбиття матриці на рядки

Базова підзадача для виконання обчислення повинна містити рядок матриці А і копію вектора b. Після завершення обчислень кожна базова підзадача буде містити один з елементів вектора результату с. Для об'єднання результатів розрахунків і отримання повного вектора з на кожному з процесорів обчислювальної системи необхідно виконати операцію узагальненого збору даних.

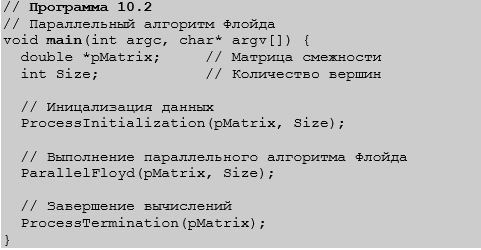


Якщо число процесорів p менше числа базових підзадач m, базові підзадачі можуть бути укрупнені з тим, щоб кожен процесор виконував кілька операцій множення рядків матриці А і вектора b. У цьому випадку, після закінчення обчислень кожна базова подзадача буде містити набір елементів результуючого вектора с.

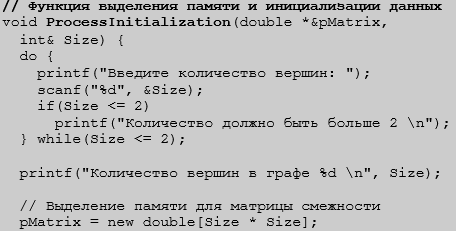
Розподіл підзадач між процесорами обчислювальної системи може бути виконане з урахуванням можливості ефективного виконання операції збору даних.

33. Використовуючи наведений програмний код, виконайте реалізацію паралельного алгоритму Флойда. Проведіть обчислювальні експерименти. Побудуйте теоретичні оцінки з урахуванням параметрів використовуваної обчислювальної системи. Порівняйте отримані оцінки з експериментальними даними.

1. Головна функція програми. Реалізує логіку роботи алгоритму, послідовно викликає необхідні підпрограми.



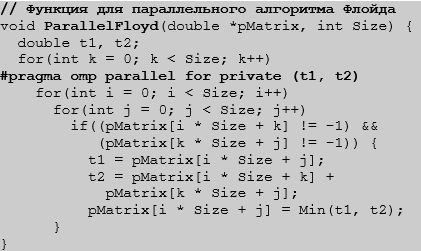
2. Функція ProcessInitialization. Ця функція призначена для ініціалізації всіх змінних, які використовуються в програмі, зокрема, для введення кількості вершин у графі, виділення пам'яті для зберігання матриці суміжності і для заповнення цієї матриці значеннями. Початкові значення елементів матриці суміжності задаються в функції RandomDataInitialization.



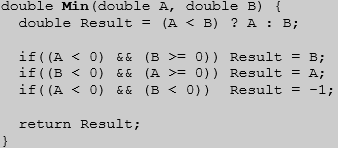


Реалізація функції RandomDataInitialization пропонується для самостійного виконання. Вихідні дані можуть бути введені з клавіатури, прочитані з файлу або згенеровані за допомогою датчика випадкових чисел.

3. Функція ParallelFloyd. Ця функція виконує паралельний алгоритм Флойда пошуку найкоротших шляхів для всіх пар вершин.



4. Функція Min. Функція Min обчислює найменше з двох чисел, з огляду на використовуваний метод позначення неіснуючих дуг в матриці суміжності (в розглянутій реалізації використовується значення -1).





### 34. Виконайте реалізацію паралельного алгоритму Прима. Проведіть обчислювальні експерименти. Побудуйте теоретичні оцінки з урахуванням параметрів використовуваної обчислювальної системи. Порівняйте отримані оцінки з експериментальними даними.

**Алгоритм Прима** - алгоритм побудови мінімального кістякового дерева зваженого зв'язного неорієнтованого графа.

Ітерації методу повинні виконуватися послідовно і, тим самим, не можуть бути розпаралелені. З іншого боку, дії, що виконуються на кожній ітерації алгоритму є незалежними і можуть реалізовуватися одночасно. Так, наприклад, визначення величин di може здійснюватися для кожної вершини графа окремо, знаходження дуги мінімальної ваги може бути реалізовано за каскадною схемою і т.д.

Розподіл даних між процесорами обчислювальної системи має забезпечувати незалежність незалежність перерахованих операцій алгоритму Прима. Зокрема, це може бути реалізовано, якщо кожна вершина графа розташовується на процесорі разом з усією пов'язаною з вершиною інформацією. Дотримання даного принципу призводить до того, що при рівномірному завантаженні кожен процесор Pj, 1 ≤ j ≤ p, повинен містити:

* набір вершин



* блок, що відповідає цьому набору з k величин



* вертикальну смугу матриці суміжності графа G з k сусідніх стовпців



* спільну(загальну \\переклад) частину набору Vj і формовану в процесі обчислень множину вершин VT

Базовою підзадачею в паралельному алгоритмі Прима може служити процедура обчислення блоку значень Δj для вершин Vj матриці суміжності A графа G.

Загальна схема паралельного виконання алгоритму Прима:

* визначається вершина графа, що має дугу мінімальної ваги до безлічі вершин вже включених в дерево; для такого набору вершин необхідно здійснити пошук мінімуму за величинами di, що є на кожному з процесорів, і виконати збірку отриманих значень на одному процесорі;
* номер обраної вершини передається всім процесорам;
* оновлюються набори величин di з урахуванням додавання нової вершини.

В ході паралельних обчислень між процесорами виконуються два види зв'язку - збір даних від усіх процесорів на одному і передача повідомлення від одного до всіх інших.

Загальний аналіз складності паралельного алгоритму Прима дає ідеальні показники ефективності:



На кожному кроці паралельний алгоритм вибирає вершину з мінімальною відстанню до споруджуваного дерева і коригує ваги di для кожної вершини. Кількість операцій при кожній з цих процедур обмежена числом. Тому на виконання n ітерацій алгоритм витратить наступний час (r - час на виконання елементарної операції):



Збірка даних на одному процесорі може бути виконана за ітерацій.

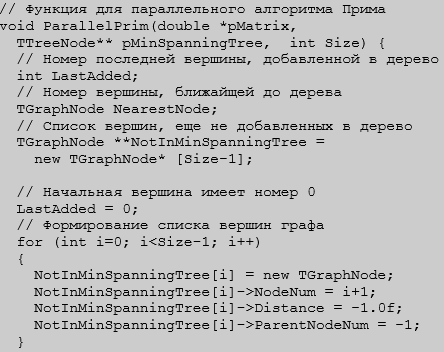
Отже, на передачу даних від всіх процесорів системи кореневому процесору буде витрачено час: де a - латентність мережі передачі даних, B - пропускна здатність мережі, w - розмір одного елемента даних в байтах.

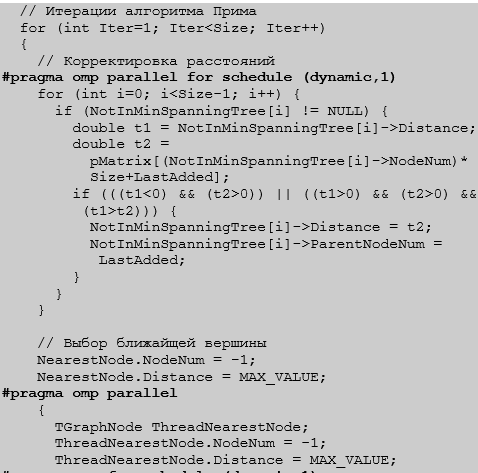
Операція розсилки одного елемента даних вимагає  операцій і може бути виконана за час 

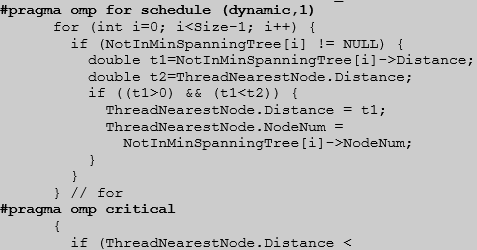
Отже, сумарний час роботи алгоритму може бути представлено співвідношенням:

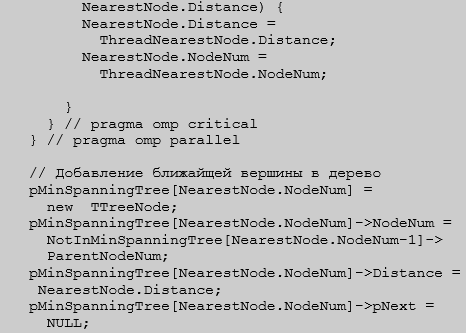


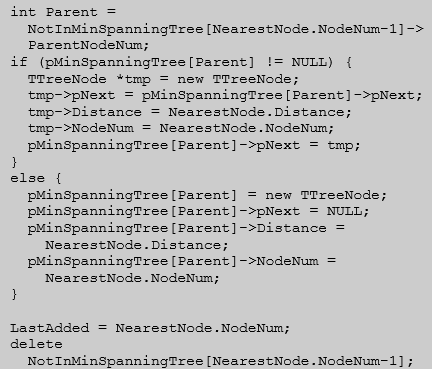
Цей вислів показує, що при великих n час роботи паралельного алгоритму буде визначатися квадратичною частиною виразу і прискорення буде близьким до ідеального. При малих n прискорення буде визначаться лінійною частиною виразу і значення прискорення буде незадовільним.

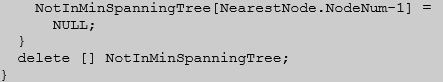












\\найадекватніше з того що я знайшла

### 35. Розробіть програмну реалізацію алгоритму Керніган - Ліна. Дайте оцінку можливості розпаралелювання цього алгоритму.

В алгоритмі Керніган-Ліна вважається, що деяке початкове розбиття графа вже існує, а потім вже наявне приближення покращується в ході декількох ітерацій. Цей спосіб покращення заключається в обміні вершинами між підмножинами наявного розбиття графа. Для формування потрібної кількості частин графа можна викоритати рекурсивну процедуру поділу навпіл.

**Алгоритм 10.3**. Общая схема алгоритма Кернигана – Лина

* Формирование множества пар вершин для перестановки. Из вершин, которые еще не были переставлены на данной итерации, формируются все возможные пары (в парах должно присутствовать по одной вершине из каждой части имеющегося разбиения графа ).
* Построение новых вариантов разбиения графа. Каждая пара, подготовленная на шаге 1, поочередно используется для обмена вершин между частями имеющегося разбиения графа для получения множества новых вариантов деления.
* Выбор лучшего варианта разбиения графа. Для сформированного на шаге 2 множества новых делений графа выбирается лучший вариант. Этот вариант далее фиксируется как новое текущее разбиение графа, а соответствующая выбранному варианту пара вершин отмечается как использованная на текущей итерации алгоритма.
* Проверка использования всех вершин. При наличии в графе вершин, еще не использованных при перестановках, выполнение итерации алгоритма снова продолжается с шага 1. Если же перебор вершин графа завершен, далее следует шаг 5.
* Выбор наилучшего варианта разбиения графа. Среди всех разбиений графа, полученных на шаге 3 проведенных итераций, выбирается (и фиксируется) наилучший вариант разбиения графа.