МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНИЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

“ Операційні системи ”

Методичні вказівки
до виконання курсового проекту
Для студентів напрямку підготовки 6.050102 «Комп’ютерна інженерія»
Професійного спрямування « Комп’ютерні системи та мережі»

Рекомендовано вченою радою факультета інформатики та обчислювальної техники НТУУ «КПІ»

Київ
НТУУ «КПІ»
2012

Операційні системи. Методичні вказівки До виконання курсової роботи
Для студентів напрямку підготовки 6.050102 « Комп'ютерна інженерія» Професійного спрямування « Комп'ютерні системи та мережі» денної та заочної форми навчання/ укладачі: В.П. Сімоненко, А.В. Сімоненко – К.: Нтуу «КПІ» ,2012. - 81с.

Гриф надано вченою радою ФІОТ
(протокол №10 від 28червня 2012 р.)

Навчальне електронне видання

" Операційні системи "
Методичні вказівки
до виконання курсової роботи
Для студентів напрямку підготовки 6.050102 «Комп'ютерна інженерія»
Професійного спрямування « Комп'ютерні системи та мережі»

Укладачі Сімоненко Валерій Павлович, д.т.н., професор
Сімоненко Андрій Валерійович, старший викладач

Відповідальний редактор: Луцький Г.М., д.т.н., проф.

Рецензент: Конвалюк Т.В., к.т.н.,доц..

За редакцією викладачів

РОЗДІЛ 1

ОСНОВИ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ АЛГОРИТМІВ СТАТИЧНОЇ ПЛАНУВАННЯ В ПАРАЛЕЛЬНИХ СИСТЕМАХ

В другому розділі було розглянуто 2 основні задачі,що розв’язуються в рамках статичного планування. Вони відносяться до класу важко розв’язуваних та NP- повних. Численні роботи, присвячені вирішенню цих задач [2, 4, 5, 10, 17, 18, 24, 32, 34, 38, 39, 112, 116], можна розділити на 2 категорії:

1. Розв’язання задачі планування;
2. Розв’язання задачі розподілу;

До першої категорії відноситься розв’язання задачі планування, що має назву “ Багатопроцесорний розклад ” [11]. Однак слід зазначити, що при мінімізації числа процесорів і / або часу рішення, як правило, не визначається в який саме обчислювальний вузол буде завантажено кожну підзадачу або задачу. Це визначається в рамках задач другої категорії [32, 34, 38, 39, 112, 116]. Об'єднання цих задач призводить до детермінованого просторово-часового плану завантаження задач в конкретне обчислювальне середовище і зазвичай використовується в системах масового розпаралелювання . Ця стратегія орієнтована на вирішення задач планування для спеціалізованих обчислювальних систем, де питання використання кожного елемента ресурсів і мінімізації часу розв’язання задачі одночасно є критично важливими.

Аналіз архітектури систем розподіленої обробки даних і застосовуваних стратегій розв’язання задач планування і розподілу в них дозволяють виділити особливості, що впливають на вибір стратегії, методу і алгоритму планування:

1.Для СРОД характерне централізоване управління виконанням розкладу паралельної задачі. Дане твердження справедливе і для СМР.

2.Реалізація розподілу завдань серед ресурсів СРОД або СМР виконується динамічним планувальником РОС з використанням "довгих команд".

3.Визначення базової архітектури ОС в значній мірі визначаючої вибір алгоритму розв'язання задачі планування:

- ОС зі спільною пам'яттю.
- ОС з розподіленою пам'яттю - PARA-PC комп'ютери з UMA, NUMA доступом до пам'яті.

- ОС з передачею повідомлень через розподілений ресурс (спільна шина, зірка). - Повнозв'язні ОС, що забезпечують передачу повідомлень між ВУ одночасно. - Комбіновані ОС (в робочих станціях, що використовують кластерну організацію, об'єднують структуру зі спільною пам'яттю або повнозв'язну в кластерах з організацією обміну між кластерами по спільній шині даних).

NP-повнота задачі планування призводить до необхідності застосування наближених, евристичних методів рішення, особливо для задач великої розмірності. Можливе застосування іншої стратегії вирішення даного завдання. Використовуючи те, що вихідна інформація для складання розкладу, як правило, задана у вигляді орієнтованого ациклічного графа в ярусно-паралельній формі, стає можливим: виконати структурний аналіз графа, його декомпозицію і звести NP-повноту задачі великої розмірності до NP-повноти задач меншої розмірності , що дозволяє використовувати локальний перебір т.к. для задач "крупнозернистого планування" довжина графа значно більше його ширини. Зворотне відношення притаманне "дрібнозернистому плануванню", пов'язаному з плануванням потоку даних. Структурний аналіз графа дозволяє виявити особливості кожної підзадачі , описати їх і виділити домінантним послідовності для кожного рівня. Саме цей підхід і використаний в даній роботі.

Основні поняття і терміни для виконання аналізу:

Під статичним плануванням розуміють складання розкладу завантаження процесорів обчислювальної системи взаємозв'язаними роботами, описаними орієнтованим графом.

При цьому виникають дві основні задачі [11].

1. Пошук мінімальної кількості процесорів, необхідних для розв'язання комплексу інформаційно і по управлінню взаємозв'язаних задач за час, що не перевищує заданий або критичний.

2. Пошук плану рішення заданого комплексу інформаційно і по управлінню взаємозв'язаних завдань на заданій кількості процесорів за мінімальний час.

Вихідною інформацією для розв'язання задач статичного планування є граф розпаралеленої вихідної задачі, заданої у вигляді ациклічного зваженого орграфа. Звичайно повинна бути визначена ціль виконання завдання: що потрібно знайти: мінімальне число процесорів або час рішення відповідно до цілей 1 або 2. Для зручності аналізу вихідний граф відображається матрицею зв'язності. Для аналізу вихідного графа необхідне перетворення графа в ярусно-паралельну форму з використанням стратегії раннього планування. Для

перетворення можна використовувати алгоритм Демукрона [23], що має часову складність O[E].

Основні терміни:

Граф задачі визначається множиною G={V, U, WV, WU},

де:

$V=\{V_1,...,V_n\}$, V_i — вершина (підзадача),

i = 1,n; n — кількість вершин графу G;

$U=\{U_l,...,U_m\}$, m — кількість дуг графу G,

де: $U_l=\{V_i, V_j\}$;

l=1,m: i=1.n: j=1,n; $V_i,V_j \in V$.

$WV \rightarrow f(V)$; $WV=\{WV_1,..., WV_n\}$;

вага вершини $WV_i=f(V_i)$; $i = \overline{1, n}$;

$WU \rightarrow \psi(U)$; $WU=\{WU_1,..., WU_m\}$;

вага дуги $WU_i=\psi(U_i)$; $i = \overline{1, m}$;

вага задачі — загальна вага всіх вершин графу $WG = \sum_{i=1}^{n} WV_i$.

Рівень L — множина вершин, для яких не існує шляху між будь-якими двума вершинами, що входять в рівень L_k :

$L_k = \{V^1, \ldots, V^{n_k}\}$, де $Vi \in V$, $i = \overline{1, n_k}$;

I $\forall p,q \in \{1,..n_k\}$: $V_p, V_q \notin U$

Нехай граф G має x рівней $L_1,..,L_x$.

Тоді $L_1 \cap L_2 \cap .. \cap L_x \equiv V$.

Шлях - множина вершин, що входять в усі рівні графу, включаючи перший і останній рівень.

$PG=\{V^1,.., V^p\}$; где $V^i \in V$, $i = \overline{1, p}$;

и $V^1 \in L_1$, $V^2 \in L_2, \ldots, V^p \in L_x$.

Отже p=x.

Нехай PG має вагу $WPG = \sum_{i=1}^{p} V_i$.

Критичний шлях W_{PGcr}. Припустимо, що існує R шляхів у графі G.

$\{PG_1, PG_2, .., PG_R\}$.Тоді критичним шляхом називається шлях

W_{PGcr}, де $W_{PGcr} = \max \{W_{PG1}, .., W_{PGR}\}$,. $W_{PG_i} = \sum_{j=1}^{p_i} V^j, i = \overline{1, R}$.

Критичний час T_{cr} - час вирішення завдань до критичному шляху (сума ваги вершин, що входять до критичного шляху). $T_{cr} = W_{P_{CR}}$

Комунікаційні витрати - час, необхідний для переміщення даних з і-ої вершини в k-у вершину - це вага дуги $U_* = \{V_i, .., V_j\}$: $T_{com}^{ij} = WU_*$

Час виконання підзадачі – вершини $T_{exe}^{i} = W_{V_i}$

Вага рівня - сума ваги усіх вершин, що знаходяться на даному рівні.

$W_{L_k} = \sum_{i=1}^{n} V^i$; Де: $L_K = \{V^1, .. V^{nk}\}$, nk <n

Критична кількість процесорів P_{cr} - мінімальна кількість процесорів, при якому час виконання завдання - T_{cr}.

При кількості процесорів менше критичного досягнення критичного часу теоретично неможливе.

Загрузка графу - це процес завантаження вершин і моделювання обчислювального процесу з метою отримання плану (розкладу) завантаження процесорів роботами чи завданнями.

Тобто для графа завдання $G_{(n)}$ (що має n вершин) необхідно скласти розклад $S_{(n, y)}$ на $P_{(y)}$ процесорах:

$S_{(n,y)}=\{\{V_1,P^1\}, \{V_2,P^2\}, . . , \{V_n,P^n\}\}$, де $V_1, . . , V_n \in V$

$P^1,..., P^n \in P_{(y)}=(P_1,.., P_y)$, зазвичай y<n.

Тоді розклад S (n, y) від G (n) і P (y) матиме два основні показники:

• Сумарний час T_{exe} паралельного виконання без урахування комунікаційних витрат.

• Реальний час T_{cexe} виконання з урахуванням комунікаційних витрат.

Тоді задачі 1, 2 формулюються таким чином:

Задача 1

- Задано граф G (n) = {V, U, WV, WU}

- Визначено критичний час φ, де $\varphi = T_{cr}$

• Необхідно визначити кількість процесорів Pmin так щоб

T_{exe} або T_{cexe} (від $S_{(n,P_{min})}$ для $G_{(n)}$ тт $P_{(P_{min})}$) = φ;

Задача 2

- Задано граф G (n) = {V, U, WV, WU}.

- Задано число процесорів.

Потрібно визначити S для завантаження графу G (n) на P_{min} процесорів так, щоб T_{exe} або T_{cexe} від було мінімальним.

Як було сказано вище, вихідною інформацією є граф задачі, заданий у вигляді ациклічного орієнтованого графа (рис. 1, 2), в якому задані не тільки час виконання кожного вузла t_{exe}, а й час комунікаційних витрат t_{com}. Вихідні дані можуть бути також представлені у вигляді матриці зв'язності, де окрім визначення зв'язку між вузлами можна дати інформацію про час комунікації (рис. 3). Час виконання кожної підзадачі (вузла) задається у вигляді вектора (рис. 4).

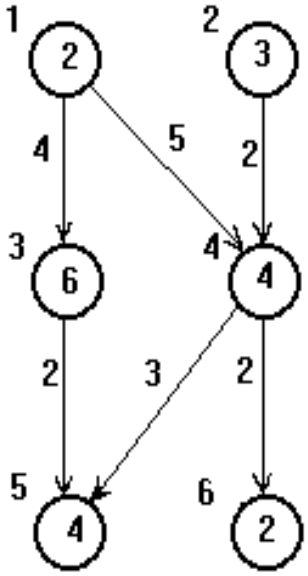


Рис.1

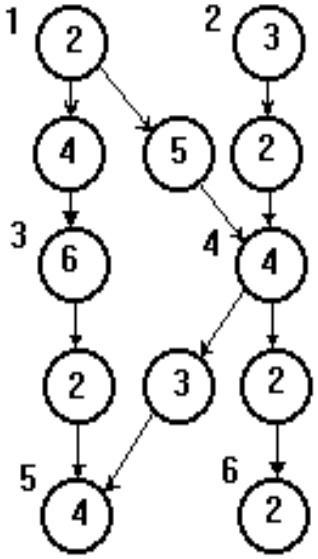


Рис.2

	1	2	3	4	5	6
1	0	0	4	5	0	0
2	0	0	0	2	0	0
3	0	0	0	0	2	0
4	0	0	0	0	0	2
5	0	0	0	0	0	0
6	0	0	0	0	0	0

Рис. 3.а

1	2
2	3
3	6
4	4
5	4
6	2

Рис. 3.б

Планирование динамического критического пути:
эффективный метод распределения задач
в мультипроцессорных системах.

Короткий огляд - У цьому розділі виконано порівняльний аналіз основних підходів до вирішення задачі багатопроцесорного розкладу, що використовують статичне планування для графів розподілу задач в повнозв'язних мультипроцесорних системах. Виконано аналіз найбільш відомих алгоритмів планування та показано, що вони мають недоліки, які можуть призвести до зниження ефективності вирішення задачі планування. Більш докладно розглянуто алгоритм, який названий алгоритмом планування динамічного критичного шляху (DCP), який відрізняється від раніше запропонованих алгоритмів кількістю розглянутих шляхів.

У ньому по-перше, визначається критичний шлях графа задачі і вибирається наступний вузол для планування в динамічному режимі. По-друге, впорядковується планування на кожному процесорі динамічно, в тому сенсі, що позиції вузлів при частковому плануванні не фіксуються поки всі вузли не будуть розглянуті. По-третє, вибирається відповідний процесор для вузла, спочатку переглянувши можливі початкові часи решти вузлів, що претендують на цей процесор, і плануються відносно менш важливі вузли на процесори що використовуються . У розділі виконано загальне порівняння для всіх аналізованих алгоритмів при змінних умовах планування. Алгоритм DCP має кращі характеристики в порівнянні з іншими алгоритмами в розглянутому діапазоні. Незважаючи на те, що алгоритм DCP володіє новими особливостями, він має допустиму тимчасову складність.

Вступ

Ефективне планування паралельної програми по процесорах є важливим для досягнення високої продуктивності паралельної обчислювальної системи. Коли структура паралельної програми: час виконання задач, залежність задач одна від одної, зв'язки завдань і синхронізація заздалегідь відомі, планування може бути виконано статично за час компіляції. Добре відомо, що мультипроцесорне планування для більшості графів задач є NP-повною задачею в загальному вигляді [12,21]. При плануванні в загальному випадку структури графу задач, що представляє програму, повинна відповідати моделям паралельних обчислювальних систем [7,14]. Однак, ця задача є NP-повною навіть для двох простих випадків: 1) планування завдань по тактам на довільній кількості процесорів [15]; 2) планування завдань за один або два такти на двох процесорах [9]. Тільки для двох особливих випадків існують оптимальні поліноміальні алгоритми. Ці випадки: планування графів завдань з однаковими обчислювальними витратами, представлених у вигляді дерева, на довільній кількості процесорів і планування довільних графів завдань з однаковими обчислювальними затратами на двох процесорах.

Однак, навіть у цих випадках, не враховуються зв'язки між задачами паралельної програми. Це означає, що планування довільного графа задач зі зв'язками всередині на два процесори є NP-повним і планування графа завдань, що мають структуру дерева з зв'язками всередині задачі на ОС з довільною кількістю процесорів також є NP-повним [25].

Паралелізм визначається структурою графа задачі, величиною розглянутої задачі, довільними обчислювальними витратами і витратами зв'язку. Більш того, для практичного використання, алгоритм планування повинен володіти малої часовою складністю і бути економічним за кількістю використовуваних процесорів. Задача планування продовжує привертати увагу дослідницької громадськості [4,5,8,13,17,18,20,23,24,27,28,29,30,32,34].В цьому розділі розглянуто алгоритм динамічного критичного шляху DCP.

Загальна постановка задачі

Паралельна програма може бути представлена направленим ациклічним графом G = (V, E), де V - множина вузлів (| V | = v) і E - множина дуг (| E | = e). Вузол в графі паралельної програми являє собою завдання, що складається з множини команд, які повинні бути виконані послідовно на одному і тому ж процесорі. З кожним вузлом зв'язана вартість обчислень, позначена як w (ni), яка показує час виконання завдання. Дуги в графі паралельної програми відповідають повідомленням зв'язку та попередніми угодами між вузлами. З кожною дугою пов'язано число, що показує час, необхідний для пересилки даних від одного вузла до іншого. Це число називається вартістю зв'язку дуги і позначається c_{ij}. Тут індекс $_{ij}$ показує, що напрямок дуги залежить від початкового вузла n_i і інцидентному йому вузла n_j. Вихідний вузол і приймаючий вузол називаються відповідно батьківським вузлом і дочірнім вузлом. У графі задачі, вузол, який не має жодного батьківського називається початковим вузлом, тоді як вузол, який не має жодного дочірнього називається кінцевим вузлом.

Ціль статичного планування - розподіл вершин графа задачі по процесорах так, що довжина планування або найкоротший шлях мінімізується без порушення попередніх обмежень. Планування вважається ефективним при малій довжині планування і прийнятній кількості використовуваних процесорів. Існує багато підходів, які використовуються при статичному плануванні. Вони включають теорію черг, теоретичні підходи побудови графів, математичне програмування і пошук простору станів [6,14]. В класичному підході [1,9], який також називається списковим плануванням, основною ідеєю є упорядкування списку вузлів шляхом присвоєння їм пріоритетів, закріплення завдань по процесорах, а потім повторення наступних двох кроків з урахуванням виконаного призначення поки не виходить план рішення.

Пріоритети визначаються статично перед початком процесу планування. В процесі планування вузол з найвищим пріоритетом є обраним для планування. На другому кроці, найкращий можливий процесор, тобто який володіє найбільш раннім початковим часом, вибирається для розміщення цього вузла.

Головною проблемою алгоритмів зі списковим плануванням є те, що призначення статичних пріоритетів не завжди впорядковує вузли планування за їх відносною значимістю. Виділення вузла, який володіє більшою значущістю, ніж інші вузли в даний момент часу призведе до істотного поліпшення планування. Недоліком статичного підходу є те, що виходить неефективне планування ,якщо вузол з відносно меншою значимістю буде обраний для планування перед вузлом з більшою значущістю. Призначення статичного пріоритету може не врахувати зміни, що відбуваються у відносній значимості вузлів протягом процесу планування. Розглянемо граф задачі, представлений на Рис. 5. Тут планування здійснюється з використанням алгоритму HLFET (найвищий рівень першим з оціночним часом), який визначає пріоритет вузла обчисленням ваги його рівня. Рівень вузла - це максимальна сума витрат обчислень уздовж шляху від вузла до кінцевого вузла. Вузол з вищим рівнем отримує вищий пріоритет. HLFET алгоритм планує вузли в порядку: n1, n2, n3, n4. Результат планування представлений на Рис.6, на якому всі вузли плануються на один процесор (PE позначений процесор); довжина планування 43 такту. Однак, довжина планування може бути зменшена, як показано на Рис. 7, ,якщо спланувати вузли в порядку: n1, n2, n3, n4. На другому кроці планування n3 є відносно більш значущим вузлом ніж n2 бо якщо його не помістити заздалегідь на процесор, початковий час n4 буде затримано завдяки збільшенню витрат зв'язку вздовж шляху n1 - n3 - n4.

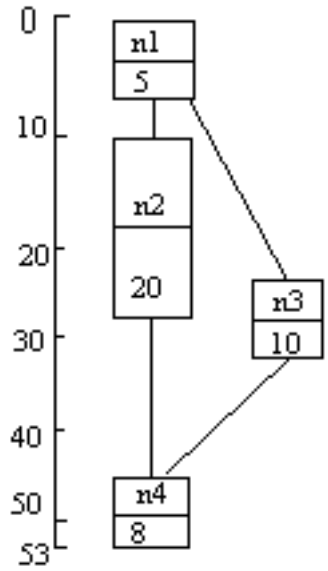


Рис. 5

Рис. 6

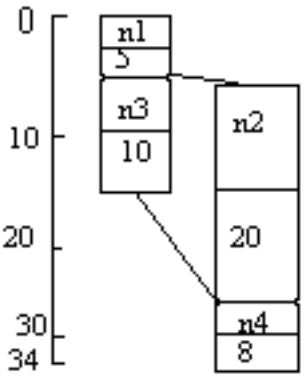


Рис. 7

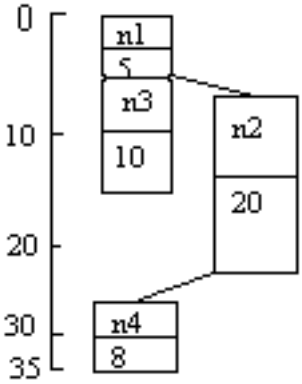


Рис. 8

Визначення 1. Критичний шлях графа задачі - це множина вузлів і дуг, що формують шлях від початкового вузла до кінцевого вузла, для якого сума витрат обчислення і витрат зв'язку максимальна.

Пріоритети вузлів перераховуються після розподілу вузлів по порядку, враховуючи зміни у відносній значимості вузлів. Таким чином, виконується повторення наступних трьох кроків в алгоритмах планування такого вигляду:

1) Визначаються нові пріоритети всіх нерозподілених вузлів.

2) Вибирається вузол з найвищим пріоритетом для планування.

3) Вибирається найбільш відповідний процесор для розміщення вузла.

Алгоритми планування, що використовують підхід, який складається з більш ніж трьох кроків, можуть здійснювати краще планування. Однак, це може збільшити складність алгоритму.

Аналіз алгоритмів статичного планування:

У цьому розділі описано шість недавно опублікованих алгоритмів планування та їх характеристики. Це алгоритм занулення вузла (EZ), алгоритм зміненого критичного шляху (MCP), алгоритм мінливого напряму (MD), алгоритм з урахуванням часу надходження завдання (ETF), алгоритм планування динамічного рівня (DLS), і алгоритм кластеризації переважаючої послідовності (DSC).

EZ алгоритм.

На відміну від алгоритмів, що основані на аналізі критичного шляху, EZ алгоритм намагається зменшити часткову довжину планування на кожному кроці, розглядаючи дугу з найбільшою вартістю в графі завдання. На кожному кроці планування, алгоритм кластеризує два вузла з дугою, що відображає найбільш складне пересилання, на один і той же процесор, не збільшуючи часткову довжину планування. Для цього EZ алгоритм спочатку складає список вершин в порядку зменшення витрат зв'язку. Потім видаляє першу вершину із списку і планує два інцидентних вузли на один і той же процесор ,якщо часткова довжина планування не збільшується. Якщо часткова довжина планування збільшується завдяки такому плануванню, два вузли плануються на різні процесори. Вузли, що знаходяться в одному і тому ж процесорі, пов'язані в порядку зменшення їх рівнів (рівні обчислюються тим же методом що і в HFLET алгоритмі). Процес повторюється, поки всі вузли не будуть розподілені. Складність EZ алгоритму буде

O(e(e+v)).

Для графа задачі, представленого на рис.5, EZ алгоритм склав план завантаження, показаний на Рис. 8. Як видно з прикладу, критерій, який використовується EZ алгоритмом для вибору вузла для планування не може як слід визначити найбільш важливий вузол на кожному кроці планування. Для цього графу задачі EZ алгоритм планує вузли в порядку: n1, n3, n4, n2. Після планування n1 і n3 вузол з найбільшою вартістю пересилки(n3, n4). Таким чином, n4 розміщується на PE 0. Однак, n2 не може бути розміщений на PE 0 пізніше, не збільшуючи довжини планування. Цей результат призведе до неефективного планування.

MCP алгоритм

MCP алгоритм розроблений з використанням атрибуту, названого як найбільш пізній можливий початковий час підключення вузла. Найбільш пізній можливий початковий час вузла визначається через (ALAP) зв'язування, яке перетинає граф задачі знизу вгору від кінцевих вузлів до початкових і виштовхує вузли зверху вниз наскільки дозволяє довжина критичного шляху. MCP алгоритм спочатку обчислює всі найбільш пізні можливі періоди для всіх вузлів. Потім, кожен вузол зв'язується зі списком найбільш пізніх початкових часів, що складаються з найбільш пізнього можливого початкового часу самого вузла, зменшуючи найбільш пізні можливі початкові часи дочірніх вузлів. MCP алгоритм складає список вузлів у порядку лексикографічного збільшення найбільш пізніх можливих початкових часів списків. На кожному кроці планування перший вузол видаляється зі списку і завантажується на процесор наскільки дозволяє найбільш ранній початковий час. MCP алгоритм спочатку був розроблений для обмеженої кількості процесорів. Складність MCP алгоритму дорівнює $O(v^2 \log v)$.

MCP алгоритм призначає більш високі пріоритети вузлів, які мають менші найбільш пізні можливі початкові часи. Однак, MCP алгоритм не обов'язково спочатку розподіляє вузли на критичному шляху. Знову розглянемо граф задачі на рис. 5. MCP алгоритм розподіляє вузли в тому ж порядку що і HLFET-алгоритм і отже план завантаження маємо той же (Рис. 6.). MCP алгоритм не може призначити точні пріоритети вузлів хоч він і враховує пересилання між вузлами для обчислення пріоритетів.

MD алгоритм

MD - алгоритм вибирає вузол на кожному кроці для планування на основі атрибуту, названого відносною мобільністю. Мобільність вузла визначається як різниця між найбільш раннім початковим часом і найбільш пізнім початковим часом вузла. Подібно ALAP зв'язуванню, згадуваному вище, найбільш ранній можливий початковий час призначається кожному вузлу через раннє (ASAP) зв'язування, яке досягається перетинанням графа задачі зверху вниз від початкових вузлів до кінцевих вузлів виштовхуючи вузли вгору наскільки це можливо. Відносну мобільність отримуємо розділивши мобільність на вартість обчислень вузла. Звичайно, вузол з нульовою мобільністю є вузлом критичного шляху. На кожному кроці MD - алгоритм завантажує вузол з найменшою мобільністю на перший процесор, який має досить великий проміжок часу для розміщення вузла без зменшення початкового часу вузла. Після завантаження вузла всі значення відносною мобільності оновлюються. Складність- $O(v^3)$.

MD - алгоритм визначає пріоритети вузлів динамічно. Хоча MD - алгоритм може правильно визначити критичний шлях вузлів для планування на кожному кроці, вибір відповідного проміжку часу і процесора не виконується як слід. Головним завданням MD - алгоритму є виштовхування планованих вузлів вниз для створення проміжку часу, необхідного для розміщення проміжного вузла без погіршення довжини планування. Може статися, що виштовхування вниз вузлів може збільшити довжину планування. Другим недоліком MD - алгоритму є те, що пошук відповідного процесора починається з першого процесора. Цей критерій вибору процесора не зменшує початкових часів вузлів на кожному кроці. Іншою проблемою MD-алгоритму є те, що вставка вузла у вільний проміжок часу відбувається без розглядання нащадків вузла. Планування, здійснюване MD - алгоритмом для графа задачі на Рис. 5 є таким же як і для EZ - алгоритму, представленому на рис.6 Коли розглядаємо вузол знаходимо проміжок часу на PE 0, достатній для розміщення цього вузла. MD - алгоритм завантажує на PE 0 не розглядаючи інші процесори. У результаті довжина планування збільшується.

ETF - алгоритм

Так само як MCP - алгоритм, ETF-алгоритм використовує статичні пріоритети вузлів і призначає їх тільки на обмежене число процесорів. Проте, вузол з найвищим пріоритетом не обов'язково планується перед вузлами з меншими пріоритетами. Це пов'язано з тим, що на кожному кроці планування ETF - алгоритм спочатку обчислює найбільш ранні початкові періоди для всіх готових вузлів і потім вибирає один з них з найменшим значенням найбільш раннього початкового часу. Вузол вважається готовим ,якщо всі батьківські вузли розподілені. Найбільш ранній початковий час вузла обчислюється перевіркою початкового часу вузла повністю на всіх процесорах. Коли два вузли мають одне і те ж значення найбільш раннього початкового часу, ETF - алгоритм завантажує вузол з більш високим статичним пріоритетом. Статичні пріоритети вузлів можуть бути обчислені на основі рівнів вузла так само як і в HLFET алгоритмі або найбільш пізніх початкових часів як в MCP алгоритмі. Складність - $O(pv^2)$, де p - кількість заданих процесорів.

Основним недоліком ETF алгоритму є неможливість зменшення часткової довжини планування на кожному кроці. Це пов'язано з тим що вузол, який має найменше значення найбільш раннього початкового часу не обов'язково знаходиться на критичному шляху. Негативним ефектом планування таких вузлів щодо вузлів критичного шляху є те, що більш ранні проміжки часу між процесорами можуть бути зайняті і, відповідно, вузли критичного шляху можуть не отримати своєчасного планування. В цьому випадку ETF - алгоритм працює таким же способом як MCP - алгоритм. Для графа задачі, представленого на рис. 5, ETF-

алгоритм здійснює таке ж планування як MCP - алгоритм рис. 6). Цього і слід було очікувати, так як обидва алгоритму в першу чергу намагаються зменшити початковий час вузла на кожному кроці.

DLS алгоритм

Подібно MD алгоритму DLS алгоритм визначає пріоритети вузлів призначенням атрибуту, названого динамічним рівнем (DL), всім нерозподіленим вузлам на кожному кроці планування. DL обчислюється використовуючи дві ознаки. Перша ознака - це статичний рівень SL (n) вузла n_i, який визначається як максимальна сума витрат обчислень уздовж шляху від n_i до кінцевого вузла. Друга ознака - це початковий час ST ($n_{i, j}$) розміщення n_i на процесорі j. Тоді динамічний рівень DL (ni, j) для пари вузол-процесор (ni, j) визначається як ST (ni) - ST (ni, j). На кожному кроці планування DLS алгоритм обчислює DL для кожного готового вузла для розміщення на процесорах. Потім вибирається пара вузол-процесор, яка містить найбільший DL серед всіх інших пар. Цей процес повторюється до тих пір поки всі вузли не будуть розміщені. Складність DLS алгоритму - $O(v^3 pf(p))$, де p - кількість заданих процесорів, а – f(p)складність алгоритму маршрутизації даних для обчислення St вузла на кожному кроці.

DLS алгоритм не призначає пріоритети на основі критичного шляху. Виконується вичерпний перебір пар[1] вузлів по процесорам на кожному кроці для знаходження вузла з найвищим пріоритетом. Ідея DLS алгоритму полягає у використанні складного параметру DL для вибору вузла з вищим статичним рівнем і меншим початковим часом планування. Однак, слід зауважити, що рівень вибраного вузла може і не бути найвищим, а його початковий час може не бути найбільш раннім серед всіх готових вузлів. Це визначає відмінність між DLS і ETF алгоритмами (зауважимо, що ETF алгоритм намагається розмістити вузол, який може починати виконання раніше і розриває зв'язки використовуючи статичні рівні). На початку процесу планування DL готових вузлів переважають над SL, тому що готові вузли знаходяться на вищих рівнях в графі завдання і їх початкові часи відповідно будуть малі. З іншого боку, коли плануємо вузли на нижніх рівнях (скажімо, кінцевих вузлів), DL готових вузлів переважають своїми початковими часами на процесорах, тому що їх SL малі, тоді як їх початкові часи великі.

Це визначає недолік в поведінці DLS алгоритму. Вузол з великим SL може бути спланований першим, навіть в тому випадку якщо його початковий час великий. Це може блокувати раннє планування більш важливих вузлів. Для графа задачі, наведеного на рис.7

DLS алгоритм здійснює планування, представлене на Рис.8 DLS алгоритм планує вузли в тому ж порядку, що і MCP алгоритм і, відповідно, отримуємо той же план.

[1] *Треба відзначити, що версії DLS алгоритму з меншою тимчасової складністю представлені в [34]. Ці версії орієнтовані на швидке виконання з меншою продуктивністю. Проте, в нашому дослідженні, ми вибрали версію, що може дати найкращу продуктивність щодо довжин планування.*

DSC - алгоритм.

DSC - алгоритм заснований на атрибуті, що має назву переважаюча послідовність, яка визначає критичний шлях частково планованого графа задачі на кожному кроці. На кожному кроці DSC - алгоритм перевіряє, чи готовий найвищий вузол критичного шляху . Якщо найвищий вузол критичного шляху готовий, DSC - алгоритм завантажує його на процесор, що дозволяє досягти мінімального початкового часу. Такий мінімальний початковий час може бути досягнуто «перерозподілом» кількох батьківських вузлів на цьому ж процесорі. З іншого боку, якщо найвищий вузол критичного шляху не готовий, то DSC - алгоритм не вибирає його для завантаження. Замість цього DSC - алгоритм обирає максимальний вузол, який лежить на шляху досяжності від критичного шляху, для завантаження. DSC - алгоритм завантажує його на процесор, що дозволяє забезпечити мінімальний початковий час вузла так, що вибір кожного процесора не затримує початковий час ще не розподілених вузлів критичного шляху.

Хоча DSC - алгоритм може визначити найбільш важливий вузол на кожному кроці планування, він не завантажує вузол, що знаходиться на критичному шляху, якщо вузол не готовий. Однак, планування з затримкою вузла критичного шляху може запобігти виникненню проміжку часу на наступних кроках планування. Іншим недоліком DSC алгоритму є те, що він використовує більше процесорів ніж необхідно, тому що завантажує вузол на новий процесор, якщо його початковий час не може бути зменшено завантаженням на будь-який процесор, що вже використовується. Однак, можна скоротити кількість процесорів завантаженням вузлів на вже використовувані процесори без збільшення довжини планування. Складність DSC алгоритму $O((e+v)\log v)$.

Для графу задачі на Рис. 5 DSC - алгоритм виконує планування,що показано на Рис. 7.

DCP алгоритм.

У цьому розділі описується DCP алгоритм планування. Як уже згадувалося раніше, хоча шість вищеописаних алгоритмів планування можуть ефективно виконати планування, кожен з

них має деякі недоліки, що впливають на якість одержуваного плану рішення. Пропонований алгоритм виключає недоліки цих алгоритмів і має такі особливості:

• Призначення динамічних пріоритетів вузлам на кожному кроці основане на динамічному критичному шляху (визначення дано нижче) таким чином, що довжина планування може монотонно зменшуватися.

• Зміна часу призначень на кожному процесорі відбувається динамічно, тобто початкові часи вузлів не фіксуються поки всі вузли не будуть розподілені.

• Вибір відповідного процесора для вузла здійснюється попередніми переглядом можливого початкового часу критичного дочірнього вузла, на цьому процесорі.

• Не здійснюється перевірка всіх процесорів для розміщення вузла. Замість цього, розглядаються тільки процесори, які містять вузли, пов'язані з даним вузлом.

• Завантаження відносно менш пріоритетних вузлів за вже використовуваними процесорами зменшує кількість використовуваних процесорів. Розглянемо деякі принципи, що використовуються в DCP алгоритмі. У першій частині обговорення описуються особливості, використовувані при виборі вузла для завантаження. У другій частині описуються критерії, які використовуються для вибору процесора в який завантажується обраний на першому етапі вузол. У таблиці 1 наведені деякі терміни та їх значення, які будуть використані в подальшому обговоренні.

Таблиця 1 Символи та їх значення

Символ	Значення
n_i	Кількість вузлів задач у графі задач паралельної програми
$w(n_i)$	Вартість обчислень для вузла n_i
c_{ij}	Вартість обчислень для дуги від вузла n_i к n_j
e	Кількість дуг в графі задачі
v	Кількість вузлів в графі задачі
CCR	Відношення пересилань до обчислення
CP	Критичний шлях графа задачі

DCP	Динамічний критичний шлях графа задачі
DCPL	Довжина динамічного критичного шляху
SL_t	Довжина планування на кроці планування t
PE(n_i)	Процесор, що містить вузол n_i
AEST (n_i, j)	Абсолютний найбільш ранній можливий початковий час завантаження n_i на процесор j
ALST (n_i, j)	Абсолютний найбільш піздній можливий початковий час завантаження n_i на процессор j

Вибір вузла

При виконанні процесу планування критичний шлях може динамічно змінюватися. Тобто, вузол, перебуваючи на критичному шляху на одному кроці, не обов'язково буде перебувати на критичному шляху на наступному кроці. Це відбувається тому, що витрати на пересилання між двома вузлами стають рівними нулю якщо вузли завантажуються на один і той же процесор. Різниця між критичним шляхом проміжного кроку планування та початковим критичним шляхом в графі задачі ми назвали динамічним критичним шляхом (DCP).

У наступній теоремі сформульовані умови для монотонного зменшення довжини планування.

Теорема 1. Нехай SL_t - середня довжина планування на кроці t процесу планування. Якщо n_i - найвищий нерозподілений вузол на DCP, початковий час якого мінімальний на кроці t, то $SL_{t+1} \le SL_t$.

Для визначення вузлів на DCP ми використовуємо два атрибути для кожного вузла: нижню межу і верхню межу початкового часу виконання вузла.

Обчислення значень цих двох атрибутів пояснюється в наступних визначеннях.

Визначення 2. Абсолютний ранній час завантаження вузла n_i на процесор j, що позначається AEST (n_i, j), рекурсивно визначається як:

$$\max_{1 \le k \le p} \left\{ AEST(n_{i_k}, PE(n_{i_k})) + w(n_{i_k}) + r(PE(n_{i_k}), j) c_{i_k i} \right\}$$

де n_i має p батьківських вузлів і n_{i_k} - k-ий батьківський вузол. AEST (n_i, j) = 0 якщо це початковий вузол, а $r(PE(n_{i_k}), j) = 1$ якщо $PE(n_{i_k}) \neq j$ і нулю в іншому випадку.

Визначення 3. Довжина динамічного критичного шляху, позначена як DCPL, визначається як: $\max_i \{AEST(n_i, PE(n_i)) + w(n_i)\}$

Визначення 4. Абсолютний найбільш пізній початковий час завантаження вузла n_i на процесор j, що позначається ALST (n_i, j) визначається як:

$$\min_{1 \le m \le q} \{AEST(n_{i_m}, PE(n_{i_m})) - r(PE(n_{i_m}), j)c_{ii_m} - w(n_i)\}$$

де n_i має q дочірніх вузлів і n_{i_m} -m- ий дочірній вузол. $ALST(n_i, j) = DCPL - w(n_i)$, якщо це початковий вузол, а $r(PE(n_{i_m}), j) = 1$ якщо $PE(n_{i_m}) \neq j$ і нулю в іншому випадку.

Зауважимо, що значення ALST обчислюються після обчислення DCPL. Обчисливши для кожного вузла AEST і ALST, вузли DCP можуть бути легко визначені.

Теорема 2. Якщо $AEST(n_i, PE(n_i)) = ALST(n_i, PE(n_i))$ тоді n_i є вузлом DCP.

За теоремою 2 можна визначити вузол DCP часто перевіряючи рівність AEST і ALST. Зменшуючи значення DCPL на кожному кроці планування, для планування вибираємо DCP - вузол, у якого немає батьківських вузлів, що знаходяться на DCP. Назвемо його найвищим вузлом DCP. Отримаємо прийнятний порядок планування DCP-вузлів, так що кожен DCP - вузол перевіряється з метою планування його після батьківського DCP-вузла.

Вибір процесора

На кожному кроці алгоритм повинен знаходити найбільш відповідний процесор, який містить найбільш відповідний для вузла час. Сформулюємо правило для управління вибором відповідного проміжку часу для надання процесора вузлу.

Правило 1. Вузол n_i може бути поміщено в процесор j, який містить послідовність вузлів, $\{n_{j_1}, n_{j_2}, \ldots, n_{j_m}\}$ якщо існує деяке « k » таке що

$$\min\{ALST(n_i, j) + w(n_i), ALST(N_{j_{k+1}}, j)\} - \max\{AEST(n_i, j), AEST(n_{j_k}, j) + w(n_{j_k})\} \ge w(n_i)$$

де $k = 0, \ldots, m$, $ALST(n_{j_{m+1}}, j) = \infty$, а $AEST(n_{j_0}, j) + w(n_{j_0}) = 0$ забезпечує, що жоден з вузлів $\{n_{j_1}, n_{j_2}, \ldots, n_{j_k}\}$ не є нащадком вузла n_i і жоден з вузлів $\{n_{j_{k+1}}, n_{j_2}, \ldots, n_{j_m}\}$ не є батьківським для n_i

Вузол не повинен бути розміщений в проміжку часу, перед яким завантажено дочірній вузол або після якого завантажений предок вузла. Зауважимо, що як тільки буде визначено

критерій для вузла, що претендує на розміщення, тобто найвищого вузла на DCP, то може статися, що не всі з батьківських вузлів розподілені.

Правило 2. Якщо вузол завантажений на процесор j, тоді

$AEST(n_i, j) = \max\left\{ AEST(n_i, j), AEST(n_{j_l}, j) + w(n_{j_l}) \right\}$ *, а*

$ALST(n_i, j) = \min\left\{ ALST(n_i, j) + w(n_i), ALST(n_{j_{l+1}}, j) \right\}$*, де l - значення k, що задовольняє правилу 1.*

Коли знайдений проміжок часу для розміщення вузла, щоб зменшити довжину DCP ми не затримуємо AEST розподілених вузлів, якщо це можливо. Тобто ми спочатку шукаємо, чи є достатньо великі проміжки часу в роботі процесора. Затримка AEST розподілених вузлів відповідає збільшенню кінцевої довжини планування, отже, кінцевий DCP може містити раніше розподілені вузли. Т.ч., коли аналізується можливість завантаження вузла на процесор, спочатку знаходиться, чи є вільний проміжок часу для примусового розміщення всіх вузлів, обмежуючи їх AEST. Якщо таких проміжків часу немає, ми ігноруємо це рішення і шукаємо інший проміжок часу.

В алгоритмі DCP не використовують стратегію простої мінімізації початкового часу. Замість цього використовується стратегія упередження початкового часу, яка виконується за наступним правилом.

Правило 3. Припустимо, що розглядається вузол n_i *для планування. Нехай* n_c *-дочірній вузол, який має найменшу різницю між AEST і ALST. Тоді* n_i *завантажується на процесор j, який дає найменше значення* $AEST(n_i, j) + ALST(n_c, j)$ *де* $ALST(n_c, j)$ *обчислюється після спроби* n_i *розмістити в j.*

При використанні правила 3 вузол може бути не розміщений в процесорі, який дозволяє досягти самого раннього початкового часу в процесі планування. Це трапляється в тому випадку, коли початкові часи дочірніх вузлів великі. Т.ч., використовуючи стратегію упередження для перевірки початкових часів критичних дочірніх вузлів, запропонований алгоритм дозволяє уникати планування вузла на невідповідний процесор. В результаті цього уникає небезпека збільшення довжини планування на наступних кроках.

Слід зазначити, що на деякому кроці планування може не виявитися жодного нерозподіленого вузла з рівними значеннями AEST і ALST. Це означає, що DCP містить тільки розподілені вузли і змін на наступних кроках планування не відбудеться. Т.ч., на наступних кроках планування немає необхідності в затримці AEST розподілених вузлів на процесорі, коли розглядається розміщення нерозподіленого вузла. Це відбувається тому, що виконавши затримку такого вузла ми не покращуємо кінцеву довжину планування. Отже, ми

можемо планувати будь-який не DCP - вузол на будь-який процесор, який може розмістити його без збільшення DCPL. Тобто ми можемо розмістити DCP - вузли на будь-якому процесорі не збільшуючи довжини планування.

Основа направленого пошуку варіанту завантаження в алгоритмі DCPC

Даний алгоритм може бути використаний для розв'язання задачі статичного планування з урахуванням і без урахування часу, що витрачається на обмін інформацією між вузлами вихідного графа завдання. При плануванні з урахуванням пересилань необхідно виконати перетворення вихідного графа із заміною пересилань додатковими вершинами графа. Таким чином після перетворення вихідного графа з пересиланнями, отримуємо зважений ациклічний граф з нульовими вагами ребер і двома видами вершин. При цьому вводиться два види критичного шляху Ср (критичний шлях без урахування пересилань) і СРС (критичний шлях з урахуванням часу пересилань по критичному шляху).

Вихідна інформація для складання плану завантаження задається у вигляді ациклічного орієнтованого графу, (А.О.Г) G(V,E,e,c)

Де:

$V = \{x_1, x_2, ..., x_n\} / \forall x_i \in V, x_i$ -- множина вершин графу G.

$$E = \{(x_1, x_2); ...; (x_i, x_j); ...; (x_m, x_n)\} / \forall (x_i, x_j) \in E \Rightarrow (x_i, x_j)$$

E – множина дуг .

$e = \{|x_1|, |x_2|, ..., |x_n|\}$ -- множина вагів вершин.

$c = |(x_i, x_j)| \forall x_i, x_j \in V.$ -- множина вагів дуг.

Введемо наступні визначення:

$Pred(x_i) = \{x_j / x_j \in V, (x_j, x_i) \in E\}.$

$Succ(x_i) = \{x_j / x_j \in V, (x_i, x_j) \in E\}.$

$In(G) = \{x_i / x_i \in V, pred(x_i) = \phi\}.$

$Out(G) = \{x_i / x_i \in V, Succ(x_i) = \phi\}.$

Вихідний граф G зображений на рис. 9

Граф, Gc (U, E ', e'), який утворений додаванням вершин нового виду це ациклічний орієнтований граф, (А.О.Г.) де:

$U = \{x_1, x_2, ..., x_n\} / \forall x_i \in U, x_i$ -- множина вершин графу G_c.

$E' = \{(x_1, x_2); ...; (x_i, x_j); ...; (x_m, x_n)\} / \forall (x_i, x_j) \in E' \Rightarrow (x_i, x_j)$ -дуга,

тобто E’ – це множина дуг між вершинами/ |E’|=2*|E| + |V|.

$e' = \{|x_1|, |x_2|, ..., |x_m|\}$ це множина вагів вершин

Цей граф зображено на рис 10

Ср - критичний шлях без урахування пересилань. (Для графа G)

СРС - критичний шлях з пересиланнями (тобто з урахуванням того, що комунікація між вершинами є теж вершиною). (Для графа Gc)

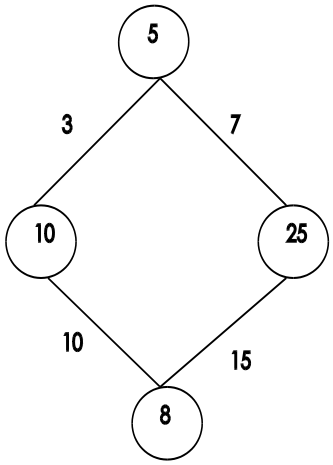


Рис. 9 вихідний граф G.

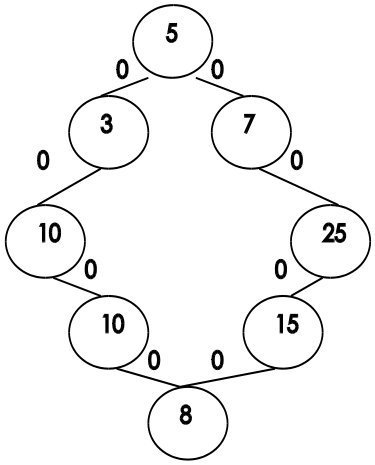


Рис. 10 .отриманий граф G_c .

T_{xi}-час розв'язання задачі x_i або вага задачі x_i.

T_c (x_i, x_j) - час комунікації між завданнями x_i і xj $\forall x_i, x_j \in G$

Для виконання дій, передбачених запропонованим алгоритмом на кожному кроці розв'язання задачі обчислення критичних шляхів та визначення критичних вершин необхідно виконати базовий, попередній розподіл задач по процесорам. При базовому розподілі передбачається, що кількість процесорів дорівнює кількості вершин (задач) вихідного графа. На рис. 11. показано базовий розподіл для вихідного графа представленого на рис. 12. При побудові базового розподілу враховуються комунікаційні витрати на передачу інформації між вершинами. При базовому розподілі передбачається, що всі пересилання виконуються повністю.

За базовим розподілом можливий просторово - часовий опис кожної вершини (задачі) та визначення $T_{b\,(xi)\,l}$, $T_{f\,(xi)\,l}$ де:

$T_{b\,(xi)\,l}$ - час початку розв'язання задачі x_i на процесорі l.

$$T_b(x_i)_l = T_b(preds(x_i)) + T_c(preds(x_i), x_i)$$

$T_f(x_i)_l$ – час закінчення задачі x_i па процесорі l.

$$T_f(x_i)_l = T_b(x_i)_l + T_{xi}$$

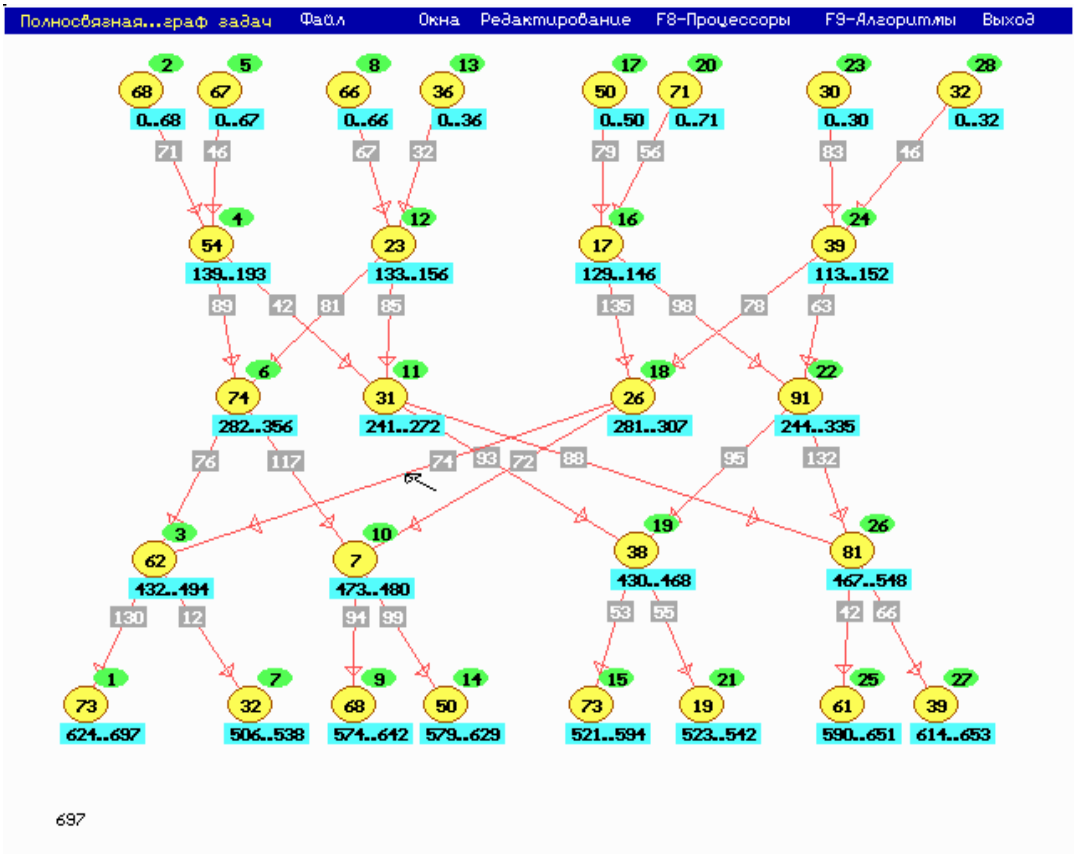


Рис.11 Вихідний граф.

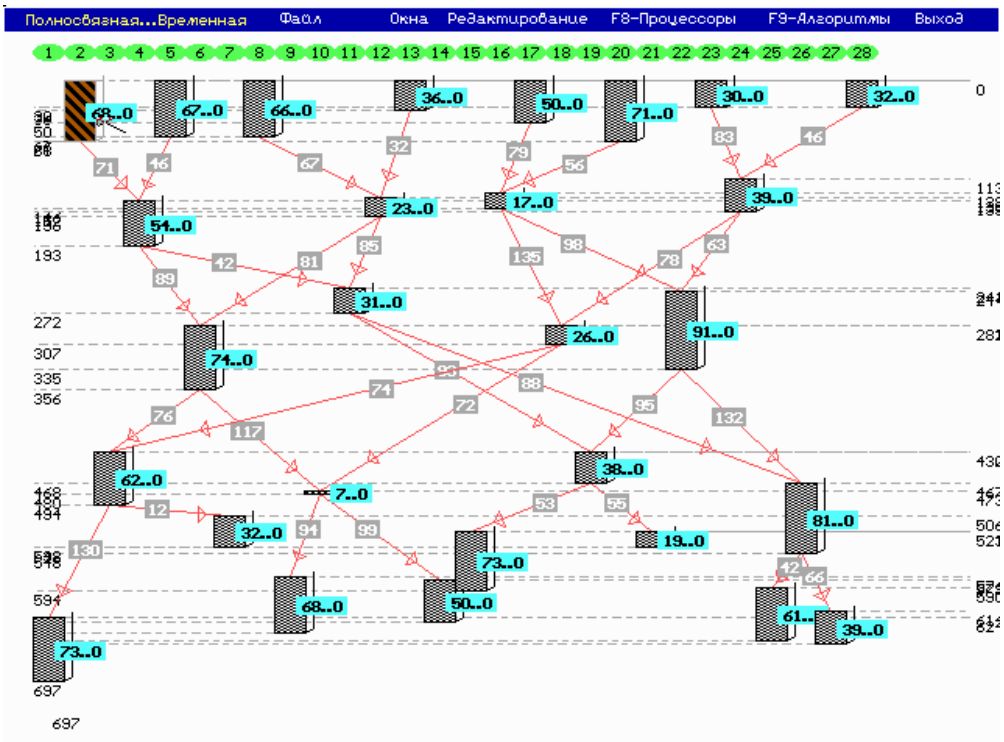


Рис. 12 Базовий розподіл.

Пошук плану оптимізованого плану завантаження виконується в декілька етапів. Першим етапом є виконання операцій пов'язаних з визначенням критичного шляху (шляхів) і завантаження вершин що входять в критичний шлях Виконання завантаження цих вершин на один процесор виконується шляхом їх кластерізації. При цьому, комунікаційні витрати між цими вершинами не враховуються , тобто відповідні вершини графа занулюються, граф редуцируется і наступний крок алгоритму виконується для нового суграфу. У кожному графі, як мінімум, є один критичний шлях (Cpc)

$$Cpc = \underset{i}{MAX} \left| \{p_1, p_2, ..., p_n\} \right| \forall x_i, x_j \in p_i \Rightarrow x_i \in E_i; x_j \in E_j \;\&\; i \neq j.$$

Після того, як визначено критичний шлях і вершини входять до нього, редуцируем граф видаляючи з графу G ці вершини, а задачі, що відповідають цим вершинам завантажуємо на один процесор. Якщо G1 = G - Cpc1, є цілим графом.

Граф G називається цілим якщо $\forall\ x_i \in G \Rightarrow x_i \in Succ\ (In(G))$.

Граф G називається не цілим якщо $\exists\ x_i \in G / x_i \notin Succ\ (In(G))$ (Рис. 13).

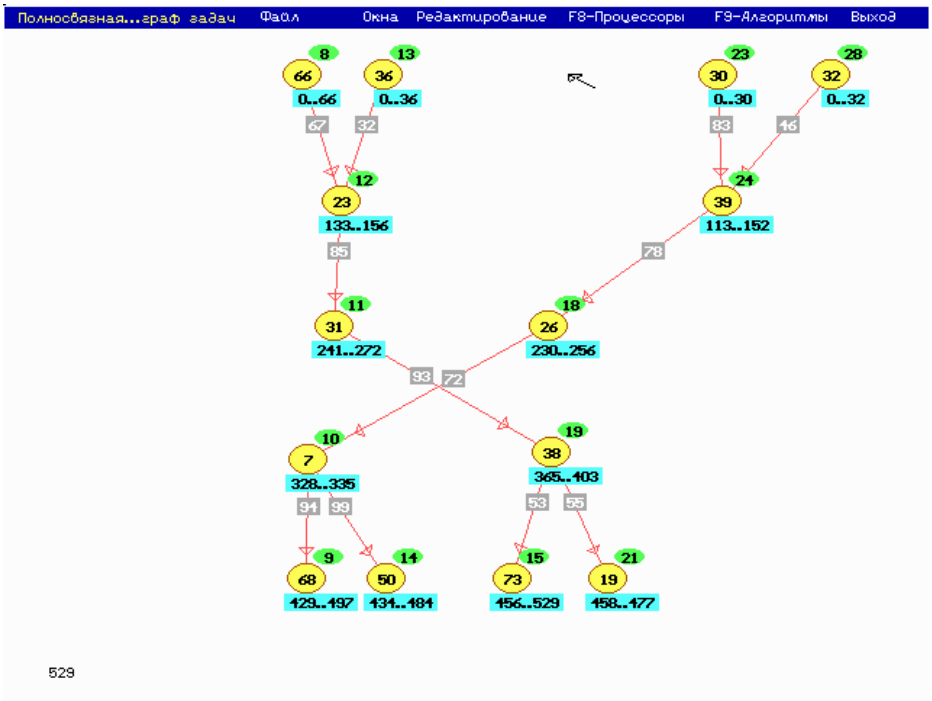


Рис. 13

Якщо G1 є графом (А.О.Г.), то продовжуємо шукати наступний критичний шлях Cpc2.до тих пір поки суграф Gk=Gk-1-Cpck є А.О.Г.

Планування виконання задач на одному процесорі призводить до обнулення часу комунікації між вершинами, що входять в критичний шлях. Таким чином завантаження всіх вершин критичного шляху на один процесор зменшує його довжину.

Твердження 1:

При завантаженні критичного шляху на один процесор ⇔ $T'_{Cpc} < T_{Cpc}$.

Твердження 2:

Нехай критичний час дорівнює TCpc, якщо ∀ $x_i, x_j \in Cpc$ / $T_f(x_i)=T_b(x_j) \Leftrightarrow$ TCpc мінімальний.

Твердження 3:

$$\forall x_i, x_j \in G; if x_i \in Cpc \,\&\, x_j \in pred(x_i) and T_f(x_j) = T_b(x_i) \Rightarrow x_j \to Cpc$$

При наявності декількох критичних шляхів, кожен завантажується на окремий процесор.

$$\forall x_i, x_{i+1} \in Cpc \Rightarrow R = | T_b(x_{i+1}) - T_f(x_i) | \geq 0$$

Якщо R = 0, то критичний шлях нерозривний й завантаження оптимальне.

Якщо R більше нуля тоді є затримка до вершини хі +1 і потрібен додатковий аналіз визначення причини затримки і можливості її видалення. Деталізація аналізу причин затримки виконується на підставі наступних правил.

Нехай $P_{ri} = \{pred(x_{i+1})\} = \{x_1, x_2, x_3, ..., x_m\}$, если $T_d(x_k)=T_b(x_{i+1})$ и $x_k \in P_{ri}$ то

то x_k- є причина затримки завдання (хі + 1). При цьому можливі три варіанти затримки.

1 – Якщо pred (x_k) Cpc тоді:

Вершина (задача), для якої виконується умова $T_{x_k} \leq | T_b(x_{i+1}) - T_f(x_i) | \Rightarrow x_k$

завантажується на той процесор, де завантажений критичний шлях .

2-Якщо pred (x_k) $\notin$ Cpc тоді можливі два варіанти:

a) завантажувати x_k на процесор де завантажений Cpc.

b) завантажувати x_k на інший процесор де t_b (x_k) мінімально.

Пояснимо їх дії на прикладі.

a) Припустимо, що t_b (x_j) і t_f (x_j) це час початок і кінця вирішенні задачі x_j (рис. 14);

$\forall x_j \in Cpc$ до завантаження x_k, і $t'_b(x_j), t'_f(x_j)$ і це час початку і кінця вирішенні задачі xj; $\forall x_j \in Cpc$ після загрузки x_k

Тоді: 1) $|t_f(x_i)-t_b(x_{i+1})| \geq t_{xk}$.

2) $t_b(x_{i+2}) - t'_b(x_{i+2}) > 0$.

Якщо виконуються умови 1) і 2) тоді x_k завантажується на процесор де Cpc, якщо ні, тоді якщо Prk = {pred (x_k)}, то необхідно додатково аналізувати P_{rk} і якщо $P_{kr} = \varnothing$ то x_k приєднується до $MAX(T_b(x_k)_j \forall x_j \in P_{rk})$. Рис. 15.

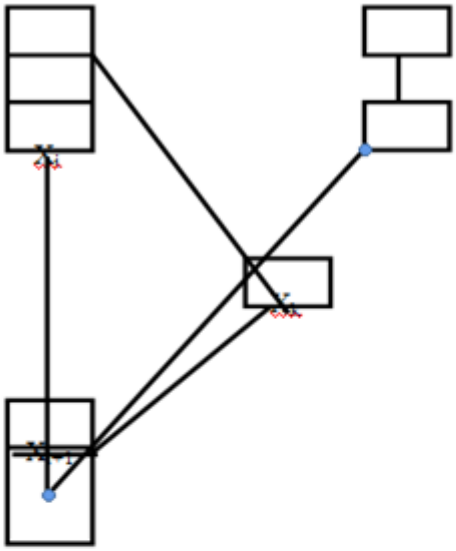


Рис.14

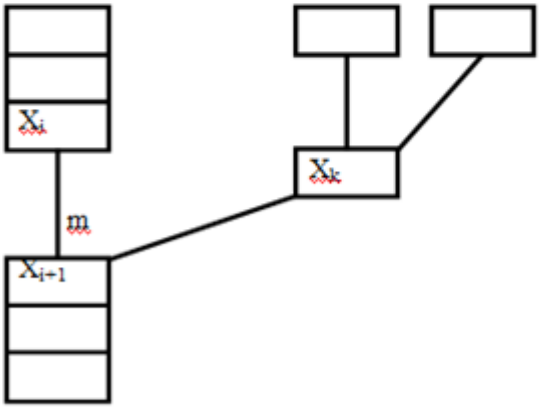


Рис.15

3 - Якщо pred (x_k) = A={ $x_1,x_2,\ldots,x_m$}/A=A_1+A_2 ; $A_1 \subset$ Cpc , $A_2 \not\subset$Cpc.

Іншими словами pred (x_k) має дві складові і одна частина з pred (x_k) належить C_{PC} а друга ні.

В цьому випадку необхідно для цієї вершини визначити pred (xk) і succ (xk). Нехай

$A_1=\{x_1,x_2,\ldots,x_i\}$, $A_2=\{y_1,y_2,\ldots,y_j\}$/pred($x_k$)=$A_1$+$A_2$;

$B_1=\{X_1,X_2,\dots X_m\}$, $B_2=\{Y_1,Y_2,\dots,Y_K\}$ /$succ(x_k)=B_1+B_2$.

В цьому випадку необхідно визначити пріоритет місця завантаження xk, за умови

$$T_{succ(x_k)} - T'_{succ(x_k)} > 0$$

Нехай

$$T_b(x_k)_i = \max \mid t_b(x_i) \mid / x_i \in A$$

$$T_f(x_j) = \max \mid t_f(x_l) \mid / x_l \in B$$

Цей випадок показано на рис.16

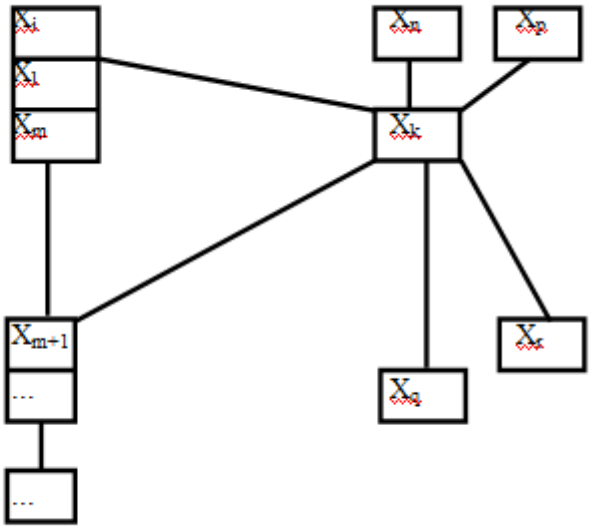


Рис. 16

У тому разі якщо:

$$T'_f(x_j) = T_f(x_k)_i + t_c(x_k, x_j) + t(x_j)$$

Тоді $$T_f(x_j) - T'_f(x_j) = \begin{cases} > 0 \Rightarrow x_k \rightarrow x_i(pred(x_k)) \\ < 0 \Rightarrow x_k \rightarrow x_j(succ(x_k)) \\ = 0 \Rightarrow x_k \rightarrow x_i или x_j \end{cases}$$

Але є особливий (специфічний) випадок коли:

$A_1\neq\{\phi\}$ і/або $B_1\neq\{\phi\}$ і $A_1\subseteq Cpc_1$, $B_1\subseteq Cpc_2$,

Тоді якщо $d_1= |T_f(x_{i1}) - T_b(x_{i1+1})|$ и $d_2= |T_f(x_{i2}) - T_b(x_{i2+1})|$; необхідно щоб виконувались умови:

1) Якщо $d_1\geq T_{xk}$ и $T_b succ(x_{i+1})-T_b succ(x_{i1+1}) >0$, то $x_k\rightarrow Cpc_1$.

2) Якщо $d_2 \geq T_{xk}$ и $T_b succ(x_{i+1}) - T_b succ(x_{i2+1}) > 0$, то $x_k \rightarrow Cpc_2$.

3) Якщо $d_i = 0$ или $d_i < T_{xk}$, то $x_k \rightarrow \max (T_b succ(x_k)_i)$.

Якщо $succ(x_i) \notin B_1 \forall x_i \in G$,тоді можливі два варіанти :

$$x_i \in Cpc \Rightarrow \begin{cases} |T_b(x_i) - T_b(x_{i+1})| \geq T_{(succ(x_i))} \Rightarrow succ(x_i) \rightarrow Cpc \\ |T_b(x_i) - T_b(x_{i+1})| < T_{(succ(x_i))} \Rightarrow succ(x_i) \end{cases}$$

завантажувати не треба

При $x_i \notin Cpc$ $succ(x_i) \rightarrow \max (T_b(succ(x_i))$.

Метою виконання описаних дій є зменшення часу рішення. Після зменшення часу рішення необхідно розглянути можливість зменшення кількості процесорів при отриманому часу рішення.

ОСНОВА СТРУКТУРНОГО АНАЛІЗУ ВИХІДНОГО ГРАФА ДЛЯ МІНІМІЗАЦІЇ ЧИСЛА ПРОЦЕСОРІВ ПРИ ПЛАНУВАННІ В ПАРАЛЕЛЬНИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ

Для решения задач планирования наиболее часто [6] используют: кластерное планирование с применением стратегий “вперед”, “назад” и планирование по критическому пути.

NP-полнота задачи планирования приводит к необходимости применения приближенных, эвристических методов решения, особенно для задач большой размерности. В данной работе предлагается другая стратегия решения данной задачи. Используя то, что исходная информация для составления расписания, как правило, задана в виде ориентированного ациклического графа в ярусно-параллельной форме, становится возможным: выполнить структурный анализ графа, его декомпозицию и свести NP-полноту задачи большой размерности к NP-полноте задач меньшей размерности, что позволяет использовать локальный перебор т.к. для задач “крупнозернистого планирования” длина графа значительно больше его ширины. Обратное отношение присуще “мелкозернистому планированию”, связанному с планированием потока данных. Структурный анализ графа позволяет выявить особенности каждой подзадачи, описать их и выделить доминатные последовательности для каждого уровня.

Следует отметить, что определение зоны поиска варианта расписания загрузки взаимосвязных задач, описанных в [5] и вычисление N_{high} является грубым и требует уточнения. Более тщательный структурный анализ исходного графа для большинства задач позволяет уменьшить значение N_{high} Вихідною інформацією для структурного аналізу є граф задачі, представлений в ярусно-паралельній формі. Кожна вершина вихідного графу після аналізу маркується і їй присвоюється ознака "транзитності" у відповідності з наступними визначеннями:

Резидентна вершина - вершина V^R належить до певного UK рівню і переміщення її на інший рівень веде до зміни критичного шляху T_{CR}.

Транзитна вершина – вершина V^{tr} має свободу вибору рівня без зміни критичного шляху TCR.

Транзитна вершина може належати трьом категоріям:

1. "явна транзитність" - зміна рівня приналежності вершини не збільшує критичний шлях;
   Умова визначення вершини "очевидної транзитності":

$$\forall X_i \in V_i \Leftrightarrow \exists X_m = succ(X_i) \in V_j \;/\; j > i+1$$

2. "неявна транзитність" - склеювання (кластеризація) вершин цієї категорії на одному рівні не збільшує критичний шлях;
   Вершина "неявної транзитності" має властивість:
   Якщо $\forall x_i \in v_i \Leftrightarrow \exists e_i = (x_j, x_i) \in E_{i-1} / x_j \in v_{i-1}; x_i \in v_i$ де

   v_i множина вершин рівня “i”

   E_{i-1} множина дуг між

   $(v_{i-1}, v_i) / \forall x_i, x_j \in v_i \Leftrightarrow T_{x_i} \geq T_{x_j}$

3. "мультиплікативна транзитність" - склеювання (кластеризація) вершин цієї категорії не збільшує критичний шлях, при цьому вершини належать різним рівням.

   - X_i^m -вершини мають властивість мультиплікативної транзитності між рівнями i і m,

якщо $\exists x_p \in V_i \;/\; T_{x_p} > \sum_{k=i}^{m} T_{x_k}$

Ступінь транзитності вершини - кількість рівнів можливого розміщення вершини K_V.

Структурний аналіз графу і вершин, що належать $\max\limits_{1 \le i \le o(x)} \{L_k\}$, дозволяє виділити VTR, що мають ознаки "очевидної", "прихованої" і "мультиплікативної" транзитності. Виділення вершин, що володіють ознакою транзитності, дозволяє виконати операцію балансування завантаженості ярусів графу задачі і зменшити його ширину. Вершини, що мають "явну" транзитність P_r^{tr}, можуть бути переміщені на інший менш завантажений рівень без зміни критичного шляху. Вершини, що мають "приховану" транзитність P_s^{tr}, можуть бути кластерізовані з іншими вершинами того ж рівня без зміни критичного шляху. Для виконання операції горизонтальної кластеризації вершин зі "прихованої" транзитного використовується евристичний алгоритм, розроблений з стратегії найбільш підходящого місця розміщення.

Використання маркування вершин графу за ознакою транзитності та виконання на його основі кластеризації дозволяє виконати побудову базового плану рішення з поліпшеними характеристиками по кількості використовуваних процесорів і їх завантаженості. Це зауваження засноване на наступному затвердження.

Твердження 1.

Кількість процесорів, необхідних для завантаження орієнтованого ациклічного графу, менше або дорівнює максимальній ширині графа.

Операція кластеризації, на основі виділення "транзитних" вершин, дозволяє мінімізувати число планованих процесорів в паралельних обчислювальних системах на етапі підготовки паралельної задачі для вирішення у розподіленій обчислювальній системі. Ці дії може виконувати планувальник "середнього" рівня. При такій схемі планування планувальник нижнього рівня виконує свої дії з урахуванням результатів структурного перетворення початкового графа.

Планувальник середнього рівня аналізує завдання представлене у вигляді графа (зваженого, ациклічного, орієнтованого), де вершина є одиницею роботи (задачею, процесом), а її вага є часом вирішення цього завдання. Дуги (гілки) визначають зв'язки між завданнями.

Ціль роботи планувальника - балансування графа.

Виконати балансування графа - це збалансувати ваги рівнів графа. Балансування виконується шляхом зміни для вершин: приладдя рівня, або їх групуванням кластеризацією).

Основна мета кластеризації або угруповання вершин зменшення ширини графу і отже, відповідно до твердження 1, зменшення кількості процесорів, необхідних для розв'язання задачі в цілому. Зменшення ширини графу, шляхом склеювання вершин, не повинно збільшувати критичний шлях.

Для цього використовуються такі типи кластеризації

1 - Горизонтальна.

2 - Вертикальна.

3 - Групова.

Горизонтальна кластеризація.

Це склеювання вершин, що мають ознаку "неявної транзитності" без збільшення критичного шляху. Вершини "неявної транзитності" володіють властивістю формула (1) і крім цього при виконанні кластеризації слід враховувати наступне:

$n_l =$ ширина графа на рівні L / $1 \le L \le M$

m = max (l) = Максимальна вага на рівні l.

$V = \sum_{k \le n_l} x_k \le m$. Де: V є новою вершиною після кластеризації вершин, що має вагу не більше max (l). Ілюстрація горизонтальної кластеризації показана на рис. 17

Для горизонтальної кластеризації виконуються наступні дії:

- всі вершини на виділеному рівні сортуються на підставі наступної умови:

$$\forall x_{ij} \in A_i \Rightarrow t_{x_{ij}} \rangle t_{x_{ij+1}} \forall i, j \in N^+ \quad (1)$$

таким чином виходить наступне розташування вершин (рис.18):

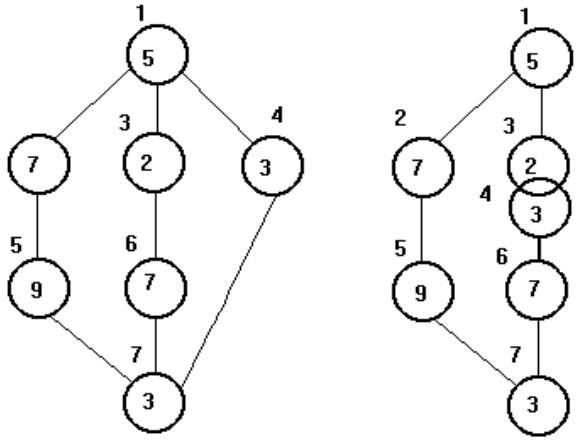


Рис. 17 Ілюстрація горизонтальної кластеризації.

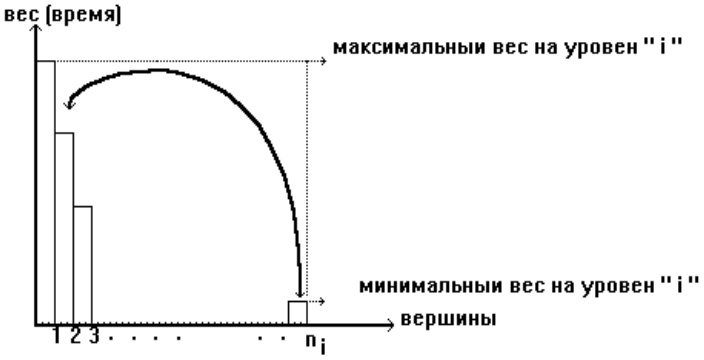


Рис .18 Початкове горизонтальне сортування вершин одного рівня.

Виходячи з прийнятої стратегії "найбільш підходящий", виконується скануюча кластеризація вершин мінімальної ваги з вершинами, що мають найменший запас часу. Ці дії виконуються рекурсивно до тих пір, поки для чергової вершини не буде жодного претендента на кластеризацію. У цьому випадку рівень збалансований.

Проілюструємо виконання горизонтальної кластеризації на прикладі.

На рис. 19 представлений вихідний, ациклічний, орієнтований граф.

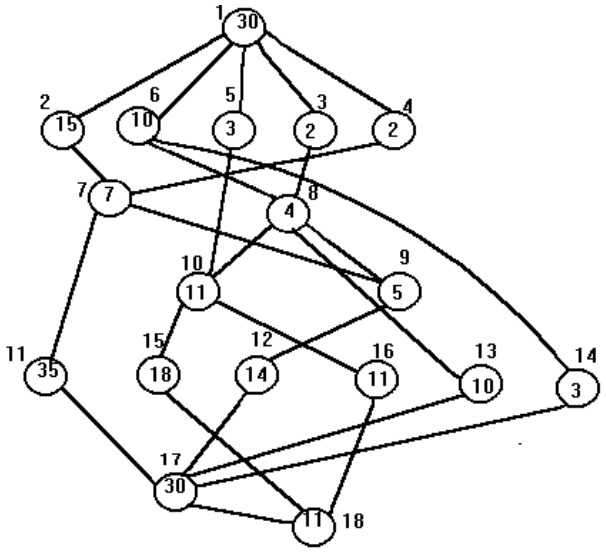


Рис. 19 Вихідний граф.

В результаті попереднього структурного аналізу графа для кластеризації виділивши критичний рівень (рівень 2), проведено сортування вершин цього рівня і кластеризація. (рис. 20). Граф, отриманий після горизонтальної кластеризації зображено на рис. 21.

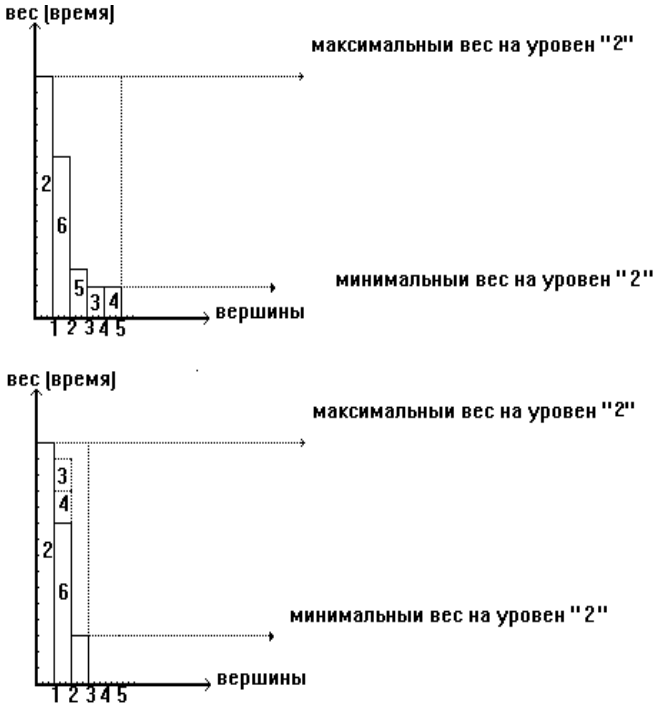


Рис. 20 Виконання процедури кластеризації для другого рівня.

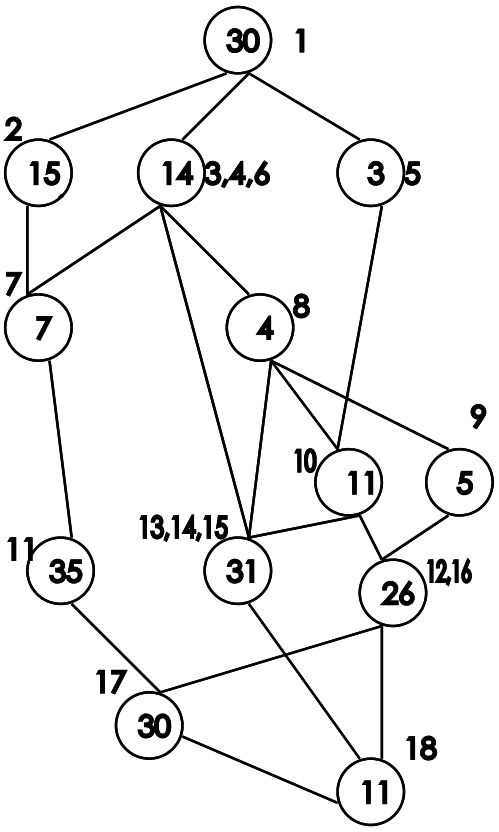


Рис. 21 Граф після горизонтальної кластеризації.

Вертикальна кластеризація:

Перенесення вершин, що відносяться до різних рівнів, з рівня на рівень не збільшує критичний шлях. Вершини які підлягають кластеризації мають наступну властивість: N = $\max\{n_1, n_2, \ldots, n_M\}$ деМ = порядкова функція графа.

n_l = ширина графа на рівні L / $1 \le L \le M$

$$\forall x_{ij} \in A_i; DES(x_{ij}) \not\subset A_{i+1} \text{ i } \left| x_{ij} + A_k \right| \le N \; / \; \forall k > i \; . .$$

На рис. 21 приведена ілюстрація вертикальної кластеризації.

3 - Групова кластеризація:

Кластеризація вершин, що належать різним рівням "i" і "m" не збільшує критичний шлях. Вершини, що підлягають груповій кластеризації мають наступну властивість:

$$x_{jk} \in V_j \; / \; i \le j \le m; DES(x_{jk}) \subset A_{m+1} \; / \; T_{x_{jk}} > \sum_{n=i}^{m} T_{x_{nk}}$$

На рис. 22 приведена ілюстрація групової кластеризації.

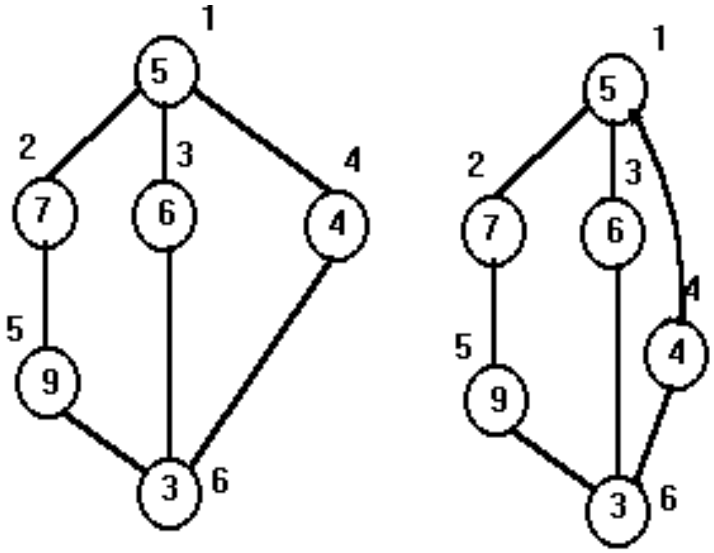


Рис .21 Вертикальна кластеризація

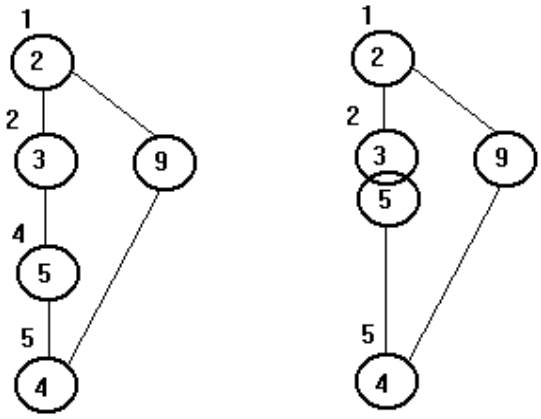


Рис. 22 Групова кластеризація.

На рис. 23 наведено результат виконання всіх трьох видів перетворення вихідного графа (рис. 18)

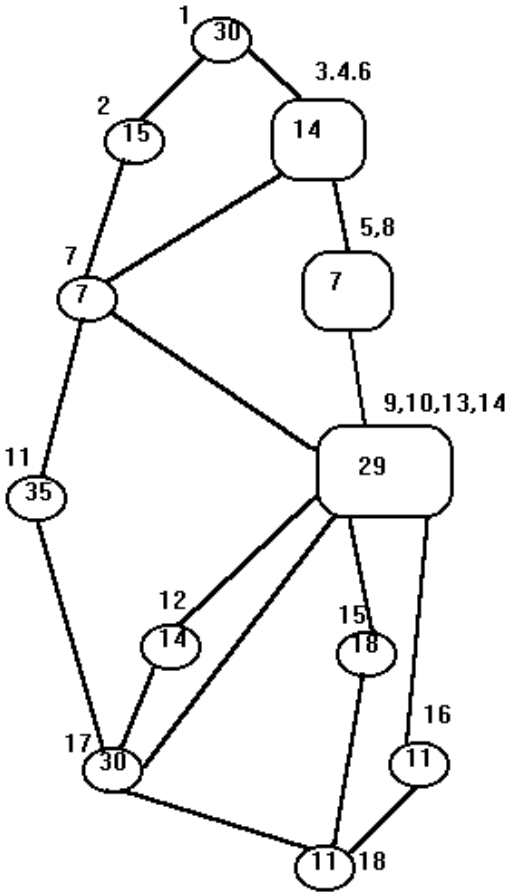


Рис. 23 Результуючий граф, отриманий після виконання всіх видів кластеризації.

ЗАСОБИ РОЗВ'ЯЗАННЯ ЗАДАЧІ СТАТИЧНОГО ПЛАНУВАННЯ В ПАРАЛЕЛЬНИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ

Скануючий просторово-часовий алгоритм планування в системах із загальною пам'яттю або UMA доступом (СПВА)

Вихідна інформація для розв'язання задачі може бути представлена у вигляді матриці зв'язності або у вигляді графа. Так як ці дві форми подання еквівалентні, то в подальшому будемо аналізувати матричну форму з використанням деяких теорем теорії графів.

Новий підхід до вирішення задачі статичного розподілу робіт в ОС із загальною пам'яттю

Ключова ідея даного підходу полягає в поділі процесу розподілу на попередній аналіз вихідної інформації, визначенні стратегії пошуку рішення і пошуку варіанту розв'язання з використанням результатів цього аналізу.

Більшість відомих алгоритмів пошуку розкладу завантаження обчислювальної системи роботами чи завданнями засновано на послідовному конструюванні варіанту розв'язання з обчисленням на кожному кроці коефіцієнтів переваги для кожного вузла (задачі), що претендує на розміщення в обчислювальному середовищі [79,124,132,135]. У цих алгоритмах передбачено повернення на попередні кроки, якщо в результаті виконання чергового кроку розміщення отримуємо неприйнятний розклад. Інші алгоритми використовують принцип генерування деякого множини варіантів з подальшим вибором найкращого, або поєднання обох підходів [67,80,90].

У запропонованому алгоритмі реалізовано новий підхід для вирішення задачі складання розкладу. Він заснований на ідеї скануючої покрокової оптимізації базового варіанту розкладу, побудованого в просторово-часових координатах без обмеження на кількість обчислювальних елементів. При такому підході стають можливими: аналіз базового варіанту розподілу, декомпозиція вихідного графа, виділення критичних зон (рівнів) і вироблення стратегії оптимізації часу або числа процесорів відповідно до функції мети.

Процес завантаження завдань в обчислювальну систему відображається в просторово-часових координатах у вигляді графіка Ганта і складається з наступних етапів:

1. Перевірка коректності завдання вихідного графа задачі.
2. Приведення графа до ярусно-паралельної форми.
3. Знаходження критичного шляху в графі.
4. Виконання попередньої оптимізації.
5. Визначення критичного числа процесорів.
6. Виконання оптимізуючого " пакування" робіт.
7. Визначення часу вирішення множини робіт при заданій кількості процесорів.

Майже завжди при плануванні має місце нерівномірний розподіл робіт за ресурсами, тобто деякі процесори завантажені повністю, деякі частково.

Логічно було б при плануванні намагатися максимально завантажувати процесори роботою, тому що завжди можливий такий розклад, при якому для його реалізації буде потрібно менше ресурсів (процесорів), ніж виділено. Вільні процесори можна використовувати для вирішення інших задач, використовувати їх для підвищення надійності системи (дублювання), або, якщо це спеціалізована система, призначена для вирішення конкретних класів задач, можна зменшити її вартість, прибравши непотрібні процесори з системи.

Перевірка коректності та приведення вихідного графа до ярусно-паралельного виду.

Перш за все будь-яким відомим алгоритмом [2] здійснюється контроль відсутності циклів в графі задачі. Приведення вихідного графа до ярусно-паралельного виду і виділення рівнів здійснюється модифікованим способом Демукрона [6] з виділенням резидентних $\mathbf{P^r}$ і транзитних $\mathbf{P^{tr}}$ вершин, обчисленням ступеня транзитності $\mathbf{K_{p^{tr}}}$ і визначенням ранніх t_p^r і пізніх термінів t_p^p закінчення виконання підзадач. Для визначення цих характеристик використаний принцип поєднання відомих стратегій раннього і пізнього планування [2]. Стратегія раннього планування використовується для визначення ранніх термінів початку робіт, а стратегія пізнього планування - для визначення пізніх термінів початку робіт. При реалізації цього етапу здійснюється скануючий розподіл заявок, готових до розв'язання, на таку кількість процесорів, яка необхідно. На відміну від відомих алгоритмів попередній (базовий) розподіл заявок на ресурси виконується при знятті обмежень на кількість процесорних елементів. Число виділених процесорів, в даному випадку, не перевищує ширину графа завдання. Тимчасова складність алгоритму побудови базового варіанту розподілу визначається дворазовим переглядом матриці зв'язності вихідного графа у відповідності зі стратегіями раннього і пізнього планування ($O[2n^2]$).

Знаходження критичного шляху в графі

Пошук критичного шляху рівносильний розв'язанню класичної задачі про пошук максимального шляху в графі. Після виконання вихідний граф представлений в ярусно-паралельній формі і пошук критичного шляху і T_{cr} для графу такого виду не викликає труднощів. Однак, за даної постановки потрібно знайти всі вершини що входять в критичні шляхи. Найбільш прийнятним для вирішення цієї задачі є алгоритм Флойда-Уоршелла, що знаходить найкоротші шляхи одночасно між усіма парами вершин і має тимчасову складність $n(n-1)^2$. У запропонованому алгоритмі поєднання стратегій раннього і пізнього планування дозволяє виділити всі вершини, що входять в критичні шляхи. Це досягається шляхом порівняння отриманих часів ранніх і пізніх термінів початку і закінчення робіт у варіантах базового розкладу двох стратегій, що й відрізняє його від аналогічного алгоритму Дейкстри. Розв'язок виходить при одному проході МС і має тимчасову складність $O(n^2)$. При скануючому розподілі заявок T_{cr} дорівнює часу виходу останньої заявки із системи. Для

визначення критичного шляху виконується перегляд попереднього розподілу і виділення задач, що входять в критичний шлях.Таким чином маємо попередній не оптимізований розклад завантаження процесорів обчислювальної системи. Кількість процесорів при такому розподілі дорівнює максимальному ступені розпаралелювання алгоритму або максимальному числу вузлів на рівнях графу (ширині графа). На рис. 24 показаний вихідний граф, на рис. 25 граф в ярусно-паралельному вигляді, а дані діаграми виконання підзадач в просторово-часових координатах після кроку базового попереднього розподілу з використанням стратегій раннього (рис. 26) і пізнього планування (рис. 27) у вигляді графіку Ганта.

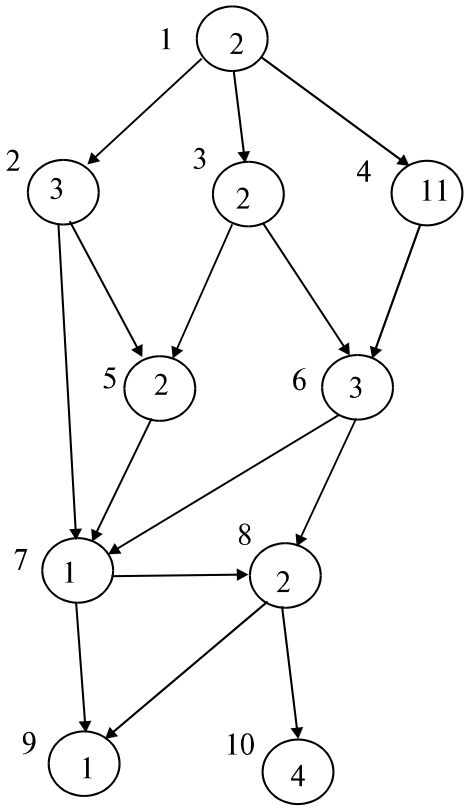


Рис. 24 Вихідний граф

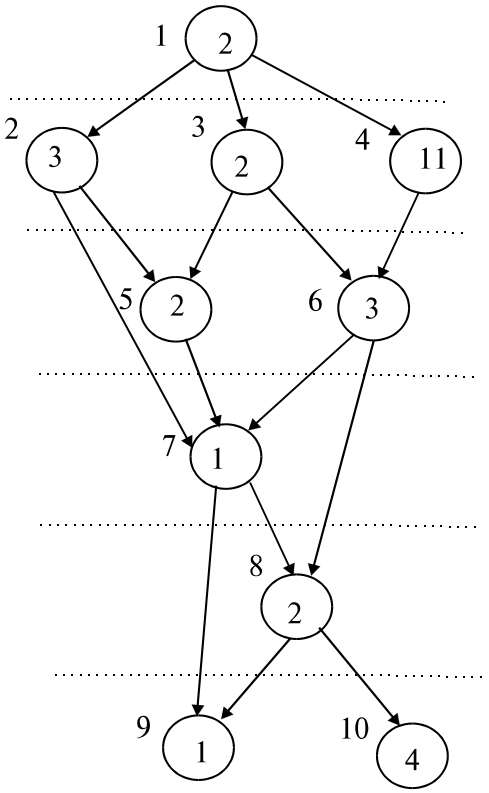


Рис. 25 Граф в ярусно-паралельній формі

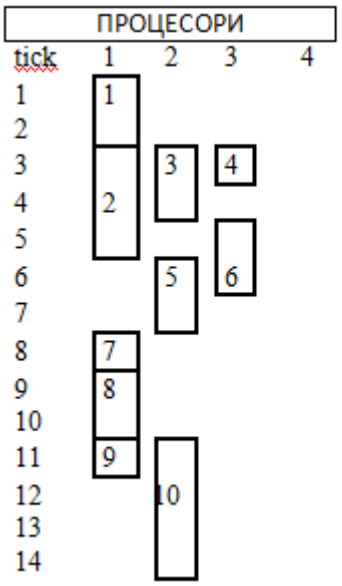


Рис. 26 Базовий розклад з використанням стратегії раннього планування

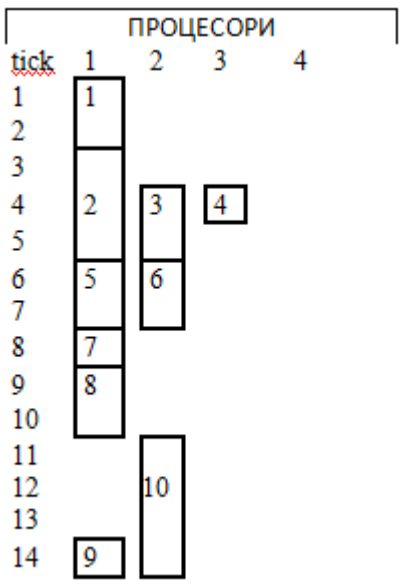


Рис. 27 Базовий розклад з використанням стратегії пізнього планування

Виконання попередньої оптимізації

Особливість алгоритму попередньої оптимізації розкладу завантаження ОС запланованими роботами обумовлена застосовуваним методом завантаження графа. Завантаження графу проводиться послідовно по рівнях, від першого до останнього. Основним критерієм при призначенні підзадачі на процесор є "нерозривність критичного шляху". Цей термін означає, що для забезпечення мінімального часу розв'язання на критичній кількості процесорів вершини, що входять в критичний шлях, повинні оброблятися на одному і тому ж процесорі. Після закінчення рахунку чергової критичної вершини процесор негайно приступить до обробки наступної вершини з критичного шляху. Вершини 1, 2, 5, 7, 8, 10 мають однакове значення початку і закінчення виконання робіт в розкладах, складених з використанням різних стратегій і є вершинами, що входять в критичний шлях (рис. 28).

Очевидно, що процесор, що обробляє вершини з критичного шляху, не може обробляти ніякі інші вершини, щоб не збільшувати час розв'язання задачі. Також очевидно, що обробка вершин критичного шляху тільки на одному процесорі призводить до відсутності пересилань даних між підзадачами, а отже і до відсутності затримок при переході від обробки однієї критичної вершини до іншої. Після перенесення задач, що знаходяться на критичному шляху, в перший процесор маємо розподіл, показаний на рис. 29. Виконання перенесення робіт, що входять в критичний шлях, в перший процесор може привести до збільшення кількості процесорів.

На першому етапі оптимізації здійснюється також переміщення підзадач з найменш завантажених процесорів в інші, без зміни рівня і початкових строків виконання робіт, визначених на попередньому кроці. При цьому перевіряється наявність "дірок" в розкладі завантаження процесорів і ці підзадачі заповнюють існуючі "дірки" (рис. 30).

Алгоритм оптимізуючого " пакування" робіт

На цьому кроці оптимізації виконується аналіз "транзитності" підзадач. Можна виділити два види "транзитності" - явна і прихована. При прихованій транзитності підзадача 4 може бути переміщена в межах свого рівня без збільшення критичного шляху . Тому на цьому кроці, в першу чергу, визначаються перезавантажені рівні, виділяються транзитні підзадачі і виконуються можливі переміщення спочатку підзадач з прихованою "транзитністю" на менш завантажені процесори цього ж рівня, а потім підзадачі з явною "транзитністю" на менш завантажені процесори інших рівнів.

Оптимізуюче пакування робіт виконується евристичним алгоритмом ОУР, розробленим на основі використання комбінації принципів: "найбільш і найменш підходящий", планування за списками з виділенням домінантних послідовностей.

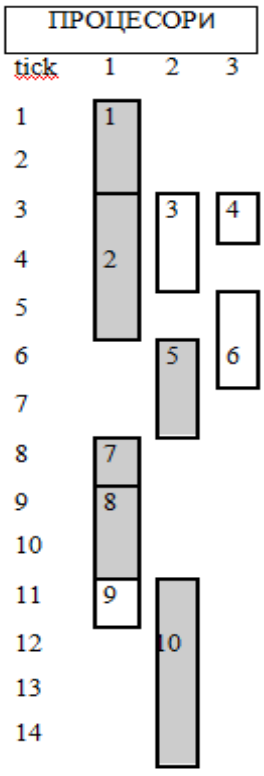


Рис. 28 Виділення вершин, що входять в критичний шлях

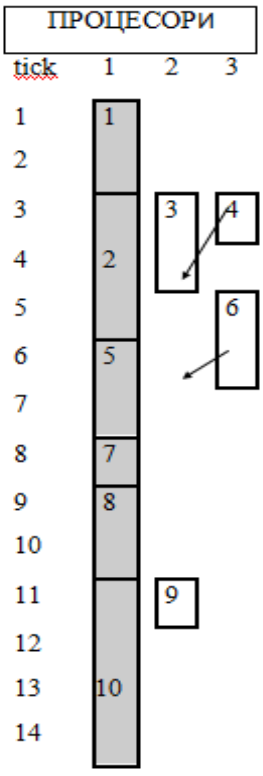


Рис. 29 Результат етапу аналізу транзитності вершин

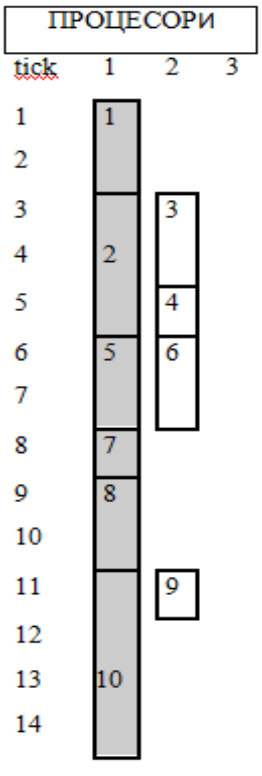


Рис. 30 Розклад після оптимізації

Опис алгоритму супроводжується прикладом.

Вихідними даними для розв'язання задачі оптимізуючого пакування є:

- Кількість процесорів (P = 4);

- Ресурс часу для кожного процесора (Tj = {12,9,7,6});

- Число задач (Z = 7);

- Ресурс необхідного часу для кожного завдання (Ti = {6,6,6,5,4,3,2}).

Поетапне вирішення завдання, для наочності будемо представляти у вигляді графіку Ганта (рис. 31).

Цільовою функцією розв'язання задачі ОУР є максимальний розподіл робіт по процесорам з мінімізацією часу простою.

Алгоритм складається з V основних кроків.

Iий крок ОУР.

Для кожної роботи обчислюється коефіцієнт запасу ресурсу Ki, j і заповнюється матриця запасу (МЗ)

$$K_{i,j} = \frac{T_j}{T_i} \text{ для } j=1..P;\ i=1..Z.$$

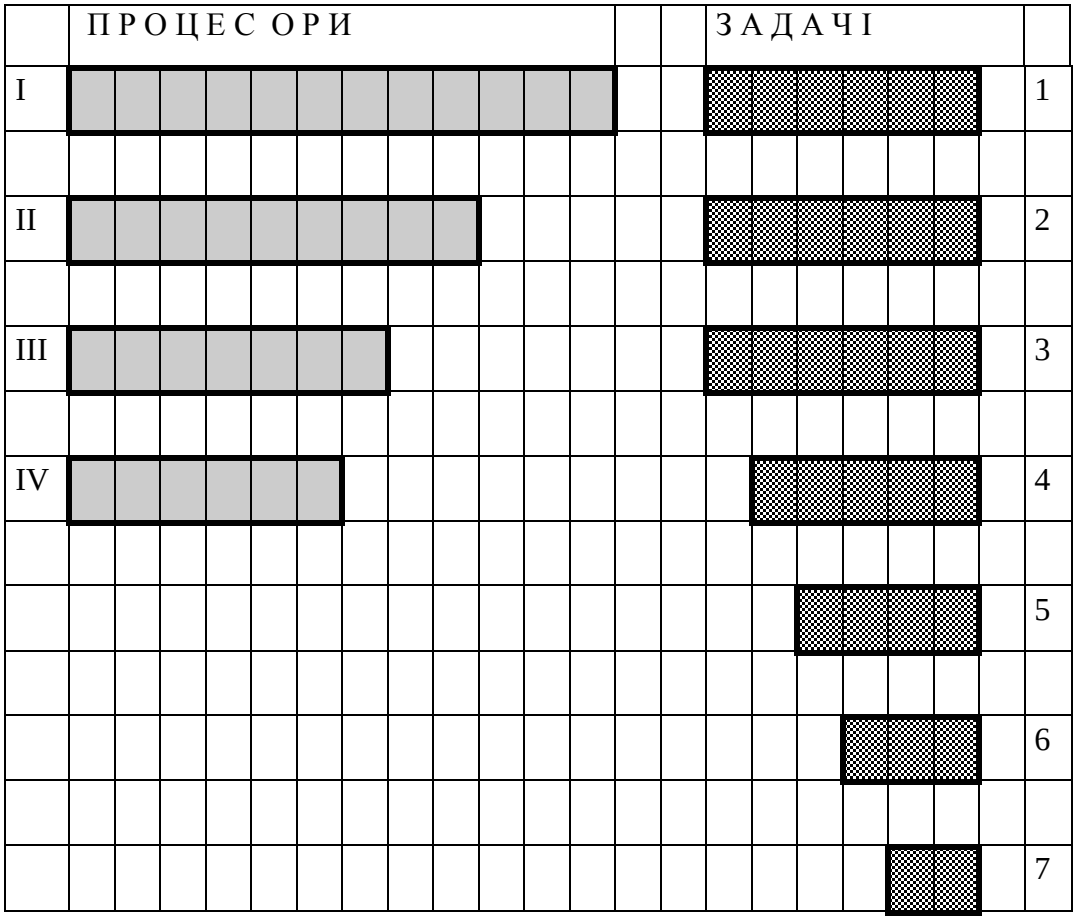


Рис. 31 Вихідний стан системи“пакування”

Якщо $K_{j,i} \geq 1$, то i-а задача може бути розміщена на j-ий процесор.

Переходимо до кроку II.

Для вихідної задачі обчислені коефіцієнти Kj, і представлені в таблиці 2.

Табл. 2 Матриця запасу (МЗ)

	1	2	3	4	5	6	7
I	2	2	2	12/5	3	4	6
II	3/2	3/2	3/2	9/5	9/4	3	9/2
III	7/6	7/6	7/6	7/5	7/4	7/3	7/2
IV	1	1	1	6/5	3/2	2	3

Крок II.

Виділяємо завдання для яких Kj, i = 1 (це означає, що час ресурсу процесора і час заявки збігаються) Тоді і-а задача, безумовно, завантажується на j-ий процесор і вони обидва виключаються з подальшого розгляду. Крок II виконується виходячи з міркувань найбільш підходящих, принципів виключає планування та покрокового конструювання.

В результаті виконання кроку II для нашого прикладу отримуємо нову МЗ (табл. 3). Перше завдання розміщується на першому процесорі і вони видаляються з розгляду.

Табл. 3 МЗ після виконання другого кроку

	1	2	3	4	5	6	7
I	0	2	2	12/5	3	4	6
II	0	3/2	3/2	9/5	9/4	3	9/2
III	0	7/6	7/6	7/5	7/4	7/3	7/2
IV	0	0	0	0	0	0	0

Якщо в скорегованій МЗ є задачі з Kj, i = 1, повторюємо крок II, інакше переходимо до кроку III.

Крок III.

З усіх Kj, i> 1 вибираємо найменший. Відповідний і-ий стовпець і j-ий рядок позначаються.

Якщо серед непомічених завдань є така, час виконання якої разом з поміченої дорівнює ресурсу часу поміченого процесора,то вони обидві завантажуються на позначений процесор і виключаються з розгляду,інакше призначення з i, j координатами вважається перспективним і маркується "×".

Якщо в МЗ є Kj, i> 0 Kj∧, ≠i "×", то повторюємо пункт III, інакше переходимо до пункту IV, знявши з усіх коефіцієнтів мітки тимчасового виключення "×".

Для прикладу К3, 2 = 7/6. Позначаємо процесор III і завдання 2. Знаходимо, що задач з Ti = $T_ш$-T2 = 1 немає.

Маркуємо К3, 2 = 7/6 знаком "×". Повторюємо пошук і маркуємо К3, 3 = 7/6 (табл. 4.3).

Повторюємо операцію знаходження min Kj, i (К3, 4 = 7/5).

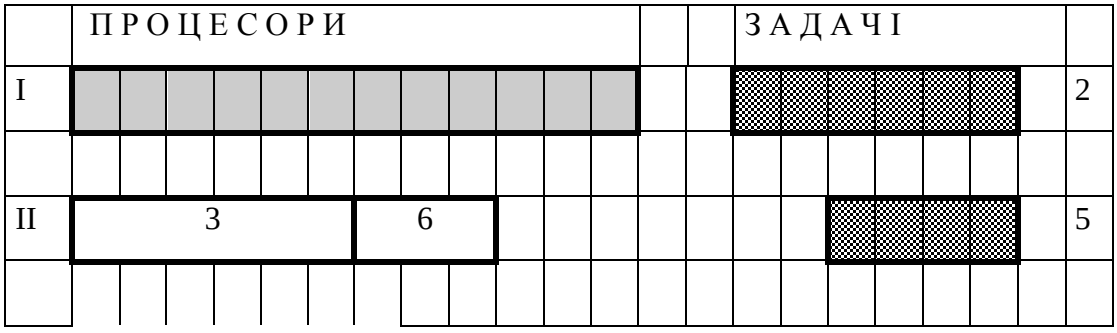
В цьому випадку $T_Ш$-T4 = 2 = T7, тобто процесор Ш може обробити задачі 4 і 7 повністю вичерпавши свій ресурс. Виконується відповідне завантажується з корекцією МЗ. Після наступного кроку визначаємо, що задачі 3 і 6 завантажуються на процесор II. Після виконання процедур I-III маємо розподіл, представлений на рис. 32 і нову МЗ в табл. 4.

Табл. 3 МЗ після маркування можливих місць розміщення

	1	2	3	4	5	6	7
I	0	2	2	12/5	3	4	6
II	0	3/2	3/2	9/5	9/4	3	9/2
III	0	×	×	7/5	7/4	7/3	7/2
IV	0	0	0	0	0	0	0

Табл. 4 МЗ після виконання кроку III

	1	2	3	4	5	6	7
I	0	2	0	0	3	0	0
II	0	0	0	0	0	0	0
III	0	0	0	0	0	0	0
IV	0	0	0	0	0	0	0



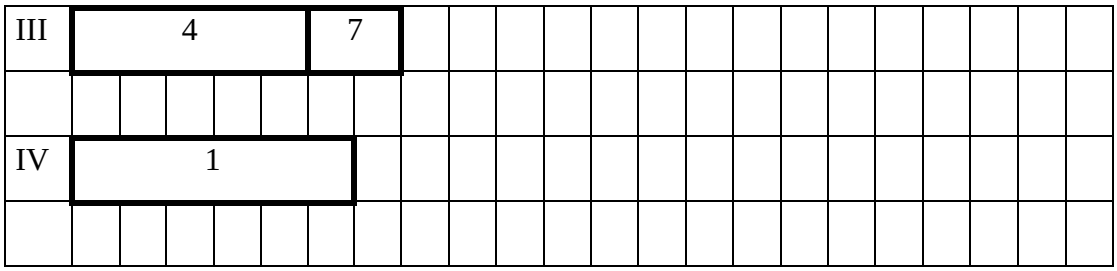


Рис. 32 Розміщення задач по процесорам після виконання кроку III

Після багаторазового виконання процедур кроку III отримуємо МЗ з Ki, j = 0 ∨ "×" Переходимо до кроку IV.

Крок IV.

З усіх Kj, i> 1 вибираємо найменше та позначаємо відповідну задачу і та процесор j.

Якщо серед непозначених задач є задача, яка разом з позначеною може бути завантажена на позначений процесор і його ресурс не вичерпаний,

то на цей процесор завантажується позначена завдання (принцип найбільш підходящий). Завдання виключається з розгляду, а ресурс процесору зменшується на величину Ti, перехід до кроку I.

інакше маркерований коефіцієнт тимчасово виключається з розгляду.

Якщо є такі Kj, i, що Kj, i> 0 Kj∧, ≠i "×", то переходимо до кроку IV, інакше переходимо до кроку V, знявши з усіх коефіцієнтів позначки винятку.

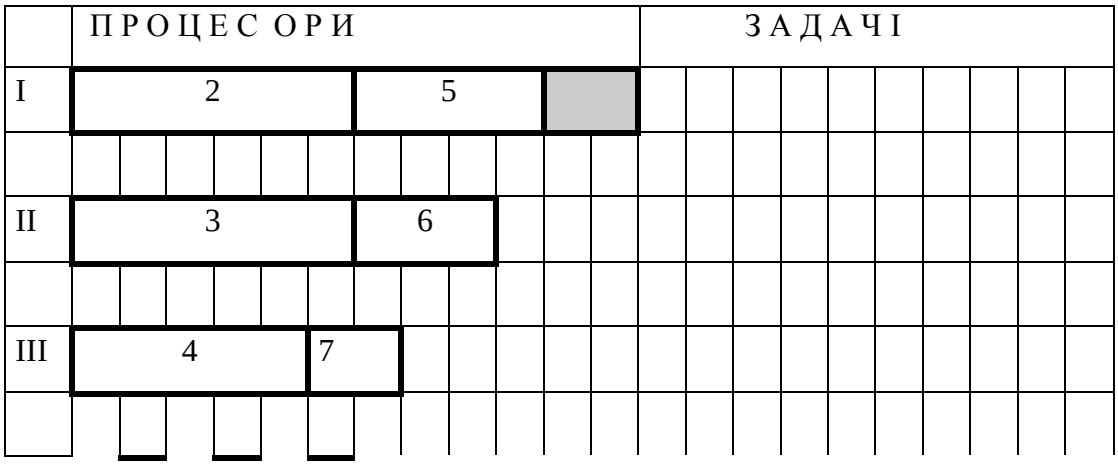
У прикладі Tp = 1 = T2 + T5 і 2-а задача завантажується на 1-ий процесор.

Виконується перерахунок коефіцієнтів і перехід до кроку V.

Крок V.

Якщо є Kj, ≥i 1, то вибирається найменший. Відповідна цього коефіцієнту завдання занурюється на процесор. Завдання виключається, ресурс процесора корегується і при наявності нерозподілених завдань виконується перехід до кроку I алгоритму.

В результаті отримуємо розподіл, показаний на рис. 33



IV	1																

Рис. 33 Остаточне розміщення задач по процесорах

Визначення часу рішення при заданому числі процесорів

Цей крок виконується тільки тоді ,коли кількість заданих процесорів менше критичного числа, отриманого на попередньому кроці. В цьому випадку потрібний покроковий перерозподіл підзадач з найменш завантажених процесорів на інші з найменшою зміною критичного часу. При виконанні цього кроку враховуються пізні терміни виконання підзадач і виконується локальний пошук місця розміщення перерозподіляємої підзадачі в рамках її "транзитності".

Алгоритм статичного планування по критичному шляху (DCPC)

Виконання дій алгоритму DCPC, описаних в розділі 3 ілюструється на прикладі пошуку плану завантаження для швидкого перетворення Фур'є. Це завдання найбільш часто зустрічається в літературі для ілюстрації працездатності алгоритмів завантаження.

З вихідного графу БПФ (Рис. 34) виділена та частина, де потрібно мінімізувати час розв'язання задачі. (Рис. 35).

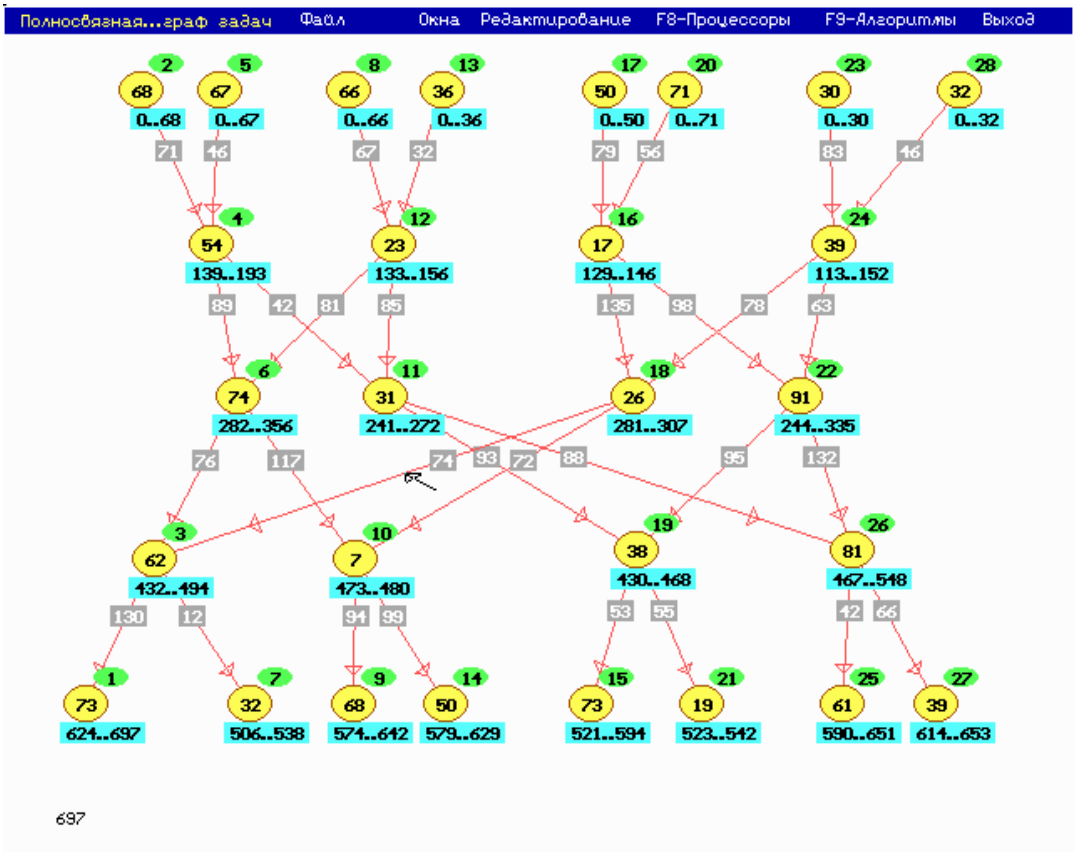


Рис.34 Вихідний граф.

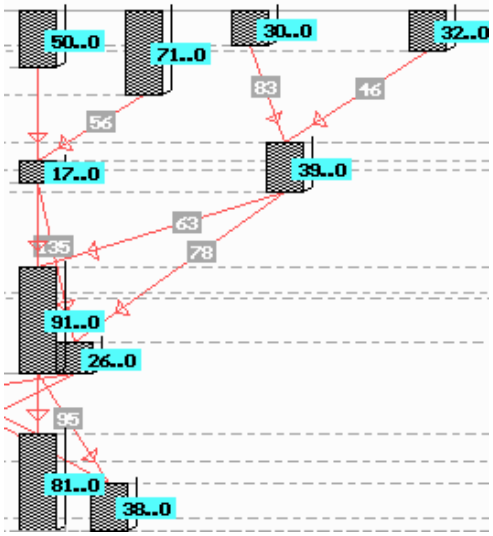


Рис. 35

Виділена частина графу має такі вершини для аналізу - Tx1 = 50; Tx2 = 71; Tx3 = 30; Tx4 = 32; Tx5 = 39; Tx6 = 17; Tx7 = 91; Tx8 = 81; Tx9 = 26; Tx10 = 7; Tx11 = 62; Tx12 = 73; Tx13 = 74; Tx14 = 91.

Вершини (x1, x6, x7, x8, ...) входять в критичний шлях цього фрагменту. Це означає, що ці вершини повинні бути завантажені на один процесор, а всі інші, тобто вершини x2, x3, x4, x5 на різних процесорах.

Якщо Tf (xi) = Tf (x1) дорівнює 50 тактів а Tb (xi +1) = Tb (x6) = 127 тактів, то це означає, що між ними є затримка, яку потрібно і / або можливо ліквідувати.

Аналізуємо можливість зменшення або ліквідації виявленої затримки. Для цього виконаємо наступні дії:

pred (x6) = {x2}; | Tf (x1)-Tb (x6) | = 77> Tx2.

Tb (xi +2) = Tb (x7) = 215.

T'b (xi +2) = T'b (x7) = 215.

Якщо Tb - T'b = 0, то x2 не впливає на мінімізацію отриманого критичного шляху і аналізується наступна вершина. У представленому варіанті | Tf (x7)-Tb (x6) | = 71 тактів, це є має місце затримка.

pred (x7) = {x5}; Tx5 = 39 <71.

Tb (xi +2) = Tb (x8) = 296.

T'b (xi +2) = T'b (x8) = 260.

Так як Tb - T'b = 36, то місцезнаходження x5 впливає на величину критичного часу, і потрібно її завантажувати на той же процесор де завантажений Cpc. Рис (36)

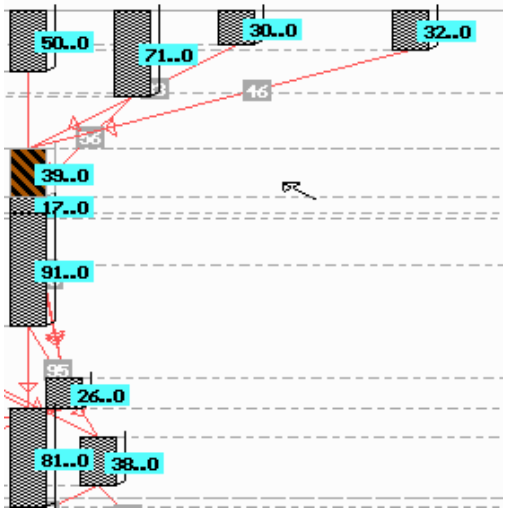


Рис. 36.

При подальшому аналізі отримуємо Tf (xi) = Tf (x1) рівне 50 тактів, а Tb (xi +1) = Tb (x5) = 113 тактів. Це означає, що між ними є затримка і її потрібно ліквідувати.

pred (x5) = {x3}; | Tf (x1)-Tb (x5) | = 63> Tx3.

Tb (xi +2) = Tb (x6) = 152.

T'b (xi +2) = T'b (x6) = 127.

Tb - T'b = 25, це означає, що x3 впливає на величину критичного часу і потрібно щоб ця вершина також може бути завантажена на той процесор, де завантажений Cpc. рис (37)

Розглянемо комбінований випадок для графу на рис. 37:

A1 = {x6, x5}; A2 = {ϕ};

B1 = {x9}; B2 = {x10}.

Tb (x9) $_{x6, x5}$ = max {279,197} = Tb (x9) x6 = 279.

Tf (x_{11}, x_{10}) = max {367,312} = Tf (x11) = 367.

T'f (x11) $_{x9}$ = 306.

Tf (x11)-T'f (x11) = 61> 0. Але так як $A_1 \in Cpc_2$ & $B_1 \in Cpc_1$,, то необхідно обчислювати d1, d2.

d_1 (x_{13}, x_{11}) = 133 ; d_2 (x_6,91) = 0. & $T_b(x_{12})$-$T'_b(x_{12})$=441-367=74>0 , $\Rightarrow$ (x_9)$\equiv$ к Cpc_1 .

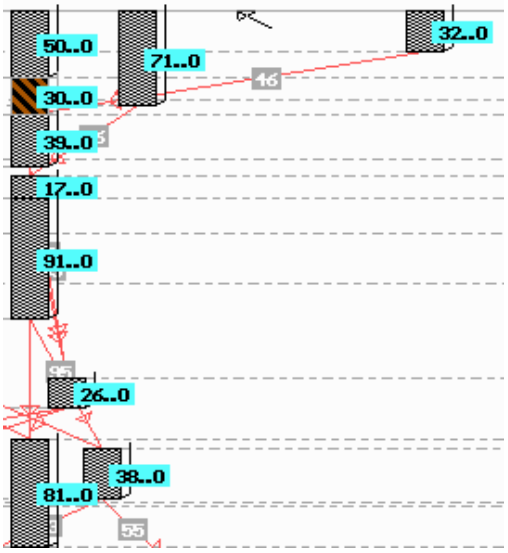


Рис.38

Після того як мінімізовано час розв'язання задачі, слід мінімізувати кількість процесорів. Перевіряємо умову можливості мінімізації

T=MAX$_i$ |T$_{Cpi}$| & $T_i = \sum_{j \le N} T_{x_j} \; / \; x_j \in Cpc_i$.

S = T-Ti;

якщо S≠0 , то ∀ x$_l$∈G/ x$_m$ ∉ Cpc$_i$; S ≥ T$_{xm}$ ⇒∃ x$_j$ ∈Cpc$_i$ / T$_f$(x$_j$)-T$_b$(x$_{j+1}$)≠0 тоді якщо T$_b$(x$_m$)≥T$_f$(x$_j$) & T'-T=0 { T' це коли x$_m$≡Cpc$_i$ }, то x$_m$ можна завантажувати на процесор де завантажений Cpc$_i$. Якщо non∃ x$_m$∈G / x$_m$≡Cpc$_i$ & T'-T=0 то:

$$\forall x_k \in P_i if \exists x_r \in P_j \; / \; x_r + P_i \; / \; T - T' = 0$$

Тоді xr потрібно завантажити на процесор Pi Рис. (39)

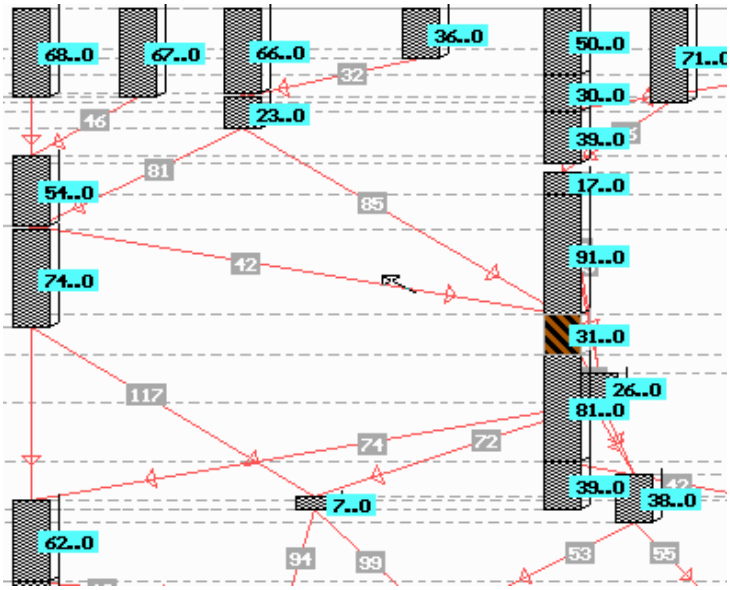


Рис.39.

Для ілюстрації запропонованого алгоритму обрано один зі складних прикладів, який був опублікований в журналі Parallel computing 20 (1994) 869-885 (Dongseung Kim, Byung-Guoen Yi; A two-pass scheduling algorithm for parallel programs). У цій стати виконано порівняння декількох алгоритмів і результати отримані при однакових вихідних даних. Автори показали, що з малими тимчасовими витратами можна отримати час планування 533 тактів, порівнюючи це рішення з оптимальним 519 тактів (simulated annealing).

Рішення запропонованого прикладу запропонованим в дисертації методом показало, що можливість завантаження запропонованого графу за 474 такти на тій же кількості процесорів.

На рис 40 показаний вихідний граф рішення задачі "Fast Fourier Transform (FFT)".

При завантаженні всіх підзадач на різних процесорах отримуємо базове рішення для вихідного графа (рис.40) з виконанням усіх пересилань між вузлами графа. Базове рішення показано на рисунку 41.

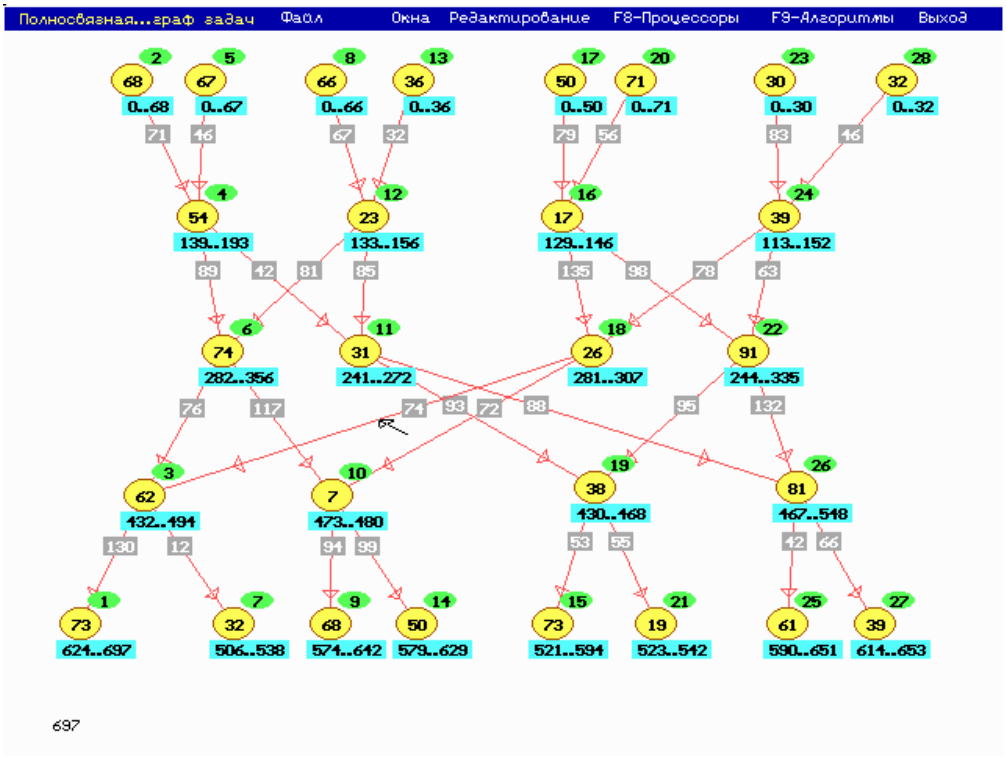


Рис. 40 Вихідний граф FFT

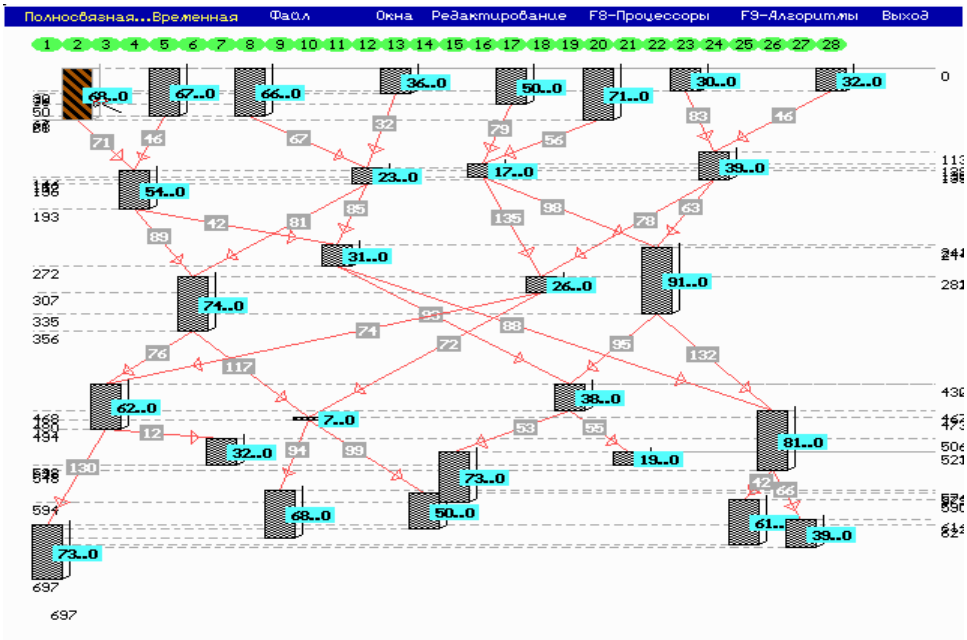


Рис. 41. Базове рішення для графу на рис (4.44.)

На малюнку 42. показаний перший критичний шлях визначений для цього графа.

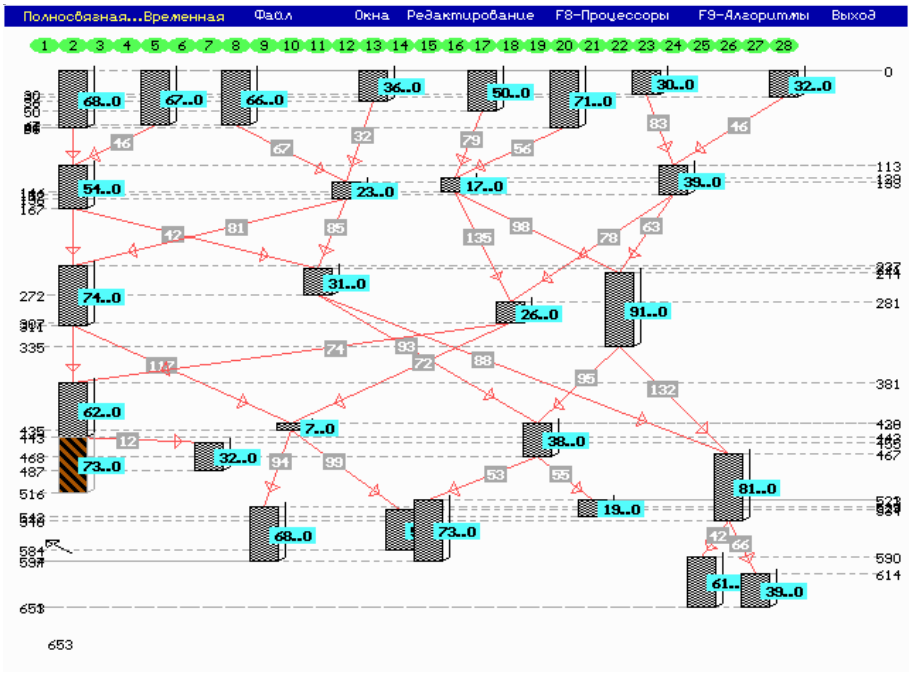


Рис. 42. Перший критичний шлях для вихідного графа.

Після того, як визначено першого критичний шлях виконується редукція графу шляхом видалення всіх вузлів критичного шляху з початкового графу "G" отримуємо новий граф, де потрібно шукати другий критичний шлях показаний на рис. 43.

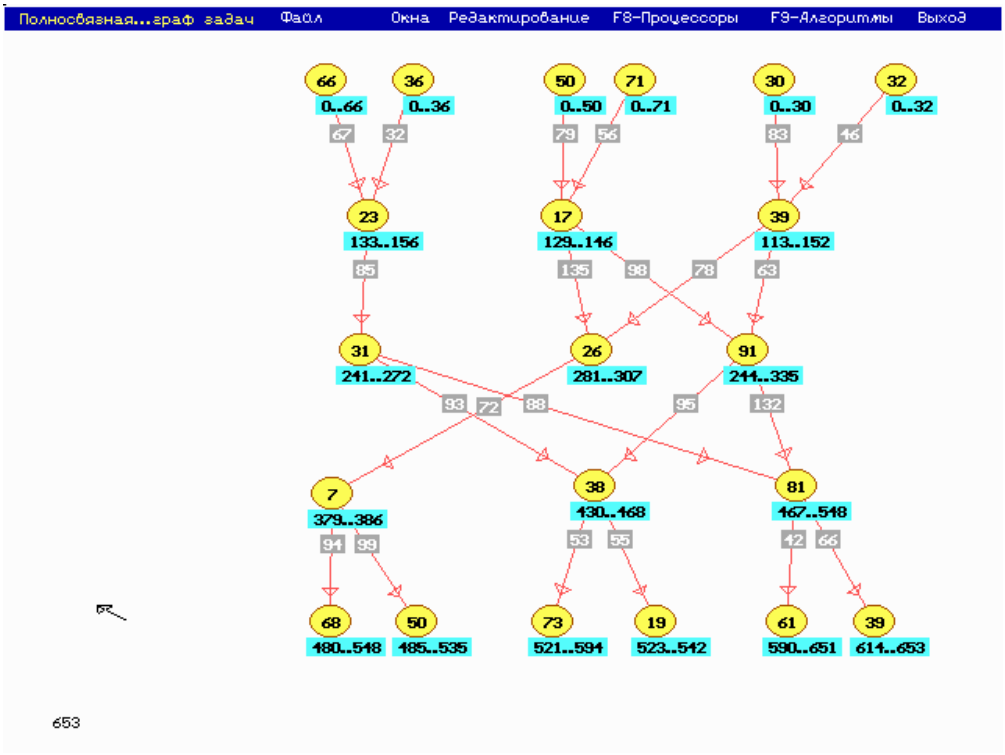


Рис. 43 Новий суграф після видалення першого критичного шляху з графу "G".

На рисунку 44. показано базове рішення для нового графу після визначення другого критичного шляху.

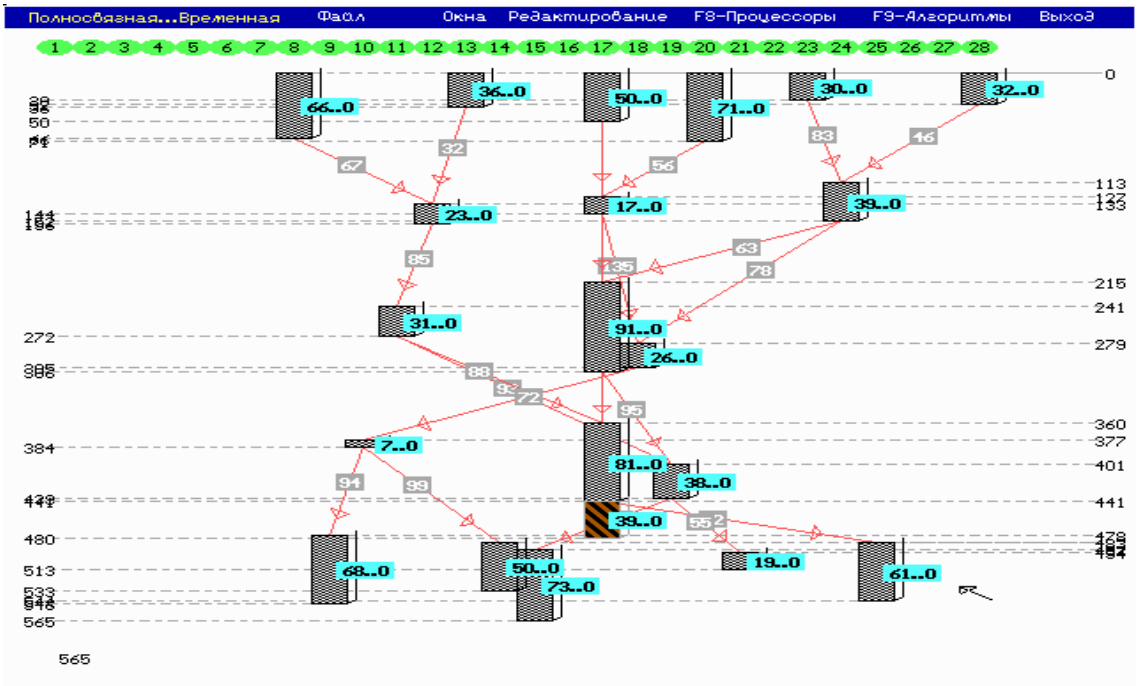


Рис. 44. базове рішення для нового графу після визначення другого критичного шляху.

На рисунку 45. показаний граф після двох кроків планування.

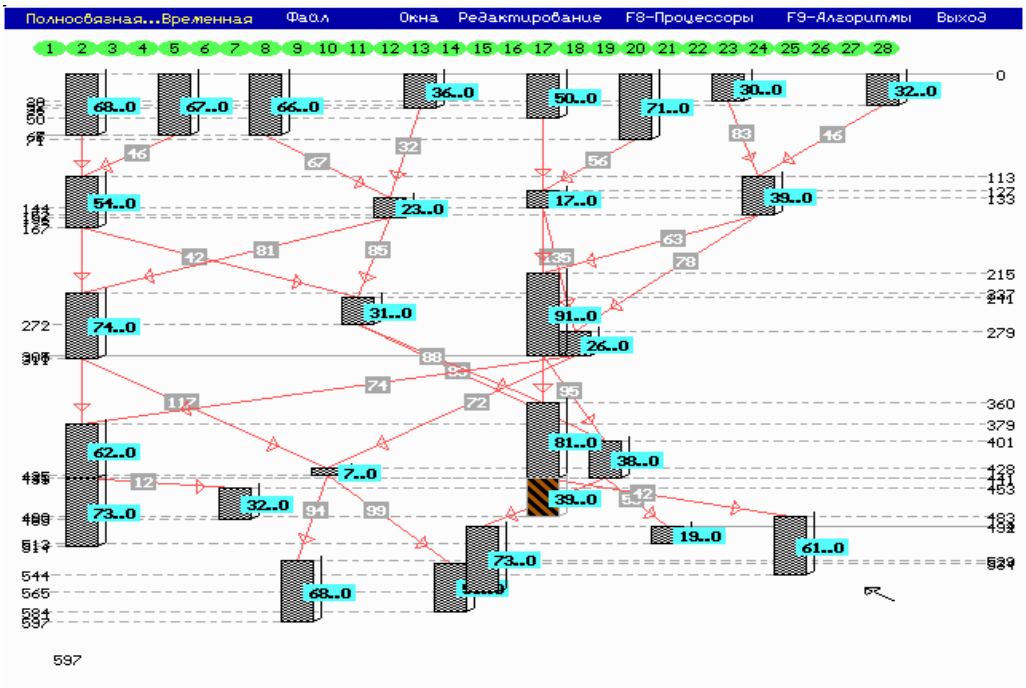


Рис. 45. Базове рішення задачі після визначеннями двох критичних шляхів.

Для початку необхідно визначити затримки між вершинами на перший критичний шлях.

$R=|T_b(X_1)-T_f(X_9)| =99<>0$ $T_b(X_9)-T'_f(X_9)= -22<0 \Rightarrow X_9$ не чіпати ..

Продовжуємо кластеризацію і так як $|T_b(X_{13})-T_f(X_9)|=70<>0 \Rightarrow$ вершина X13 має затримку і потрібен додатковий аналіз.

Припустимо, що Pr (X13) = $\{X_9, X_{10}\}$ & Td (X_{10}) = 237 = Tb (X_{13}) $\Rightarrow$ вершина X10 є причиною затримки вершини X13.

$X_{10}. \notin$ Ccp и $|T_f(X_9)-T_b(X_{13})|=70 > T_{X10}$ но pred(X_{10})=$\{X_3,X_4\} \not\subset$ Cpc$\Rightarrow X_{10} \rightarrow X_3$ или X_4.

$T(X_{10})_{X3}=66+67=133$.

$T(X_{10})_{X4}=36+32=68$.

$\Rightarrow X_{10} \rightarrow X_3$ це означатиме, що вершини X_{10}, X_3 слід завантажити на один процесор (Рис. 46)

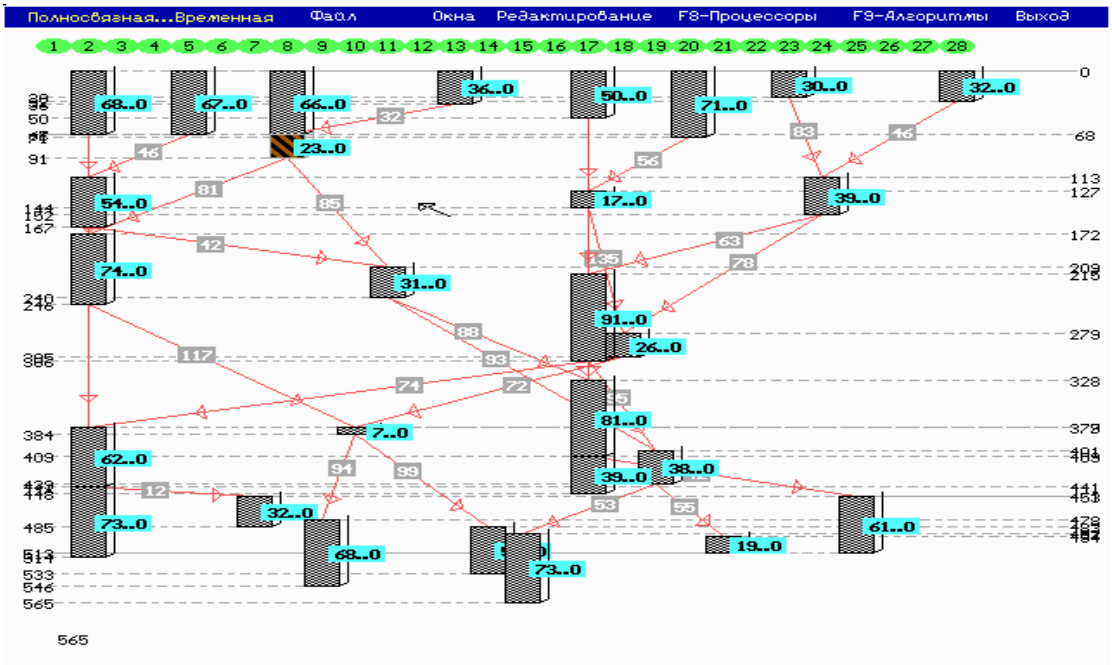


Рис. 46. Результат кластеризації вершини № 10 з вагою 23 такти.

Якщо проаналізувати Cpc2 то $R=|T_b(X_{11})-T_f(X_5)|=77 \Rightarrow$ присутня затримка вершини X11 і потрібен додатковий аналіз.

Pred (X11) = {X6} Td (X6) = 127 = Tb (X11) та Tb (X16)-T'b (X16) = 0, що означає, що кластеризація вершин X6, X11 на один процесорах не принесе бажаного результату.

Продовжуємо подальший аналіз. R = | Tb (X16)-Tf (X11) | = 71 $\Rightarrow$ присутня затримка вершини X16.

Pred (X16) = {X12} èTd (X12) = 215 = Tb (X16), означає, що вершина X12 є причиною затримки, але:

1) $R=|T_b(X_{16})-T_f(X_{11})|=71 \geq T_{X12}$ и 2) $T_b(X_{20})-T'_b(X_{20})>0$

Виконання умов 1 та 2 свідчить, що вершину X12 потрібно завантажити на процесор де завантажений Cpc_2.Ti ж умови виконуються і для вершини X7

Виконання цих дій показано на рис.47-48.

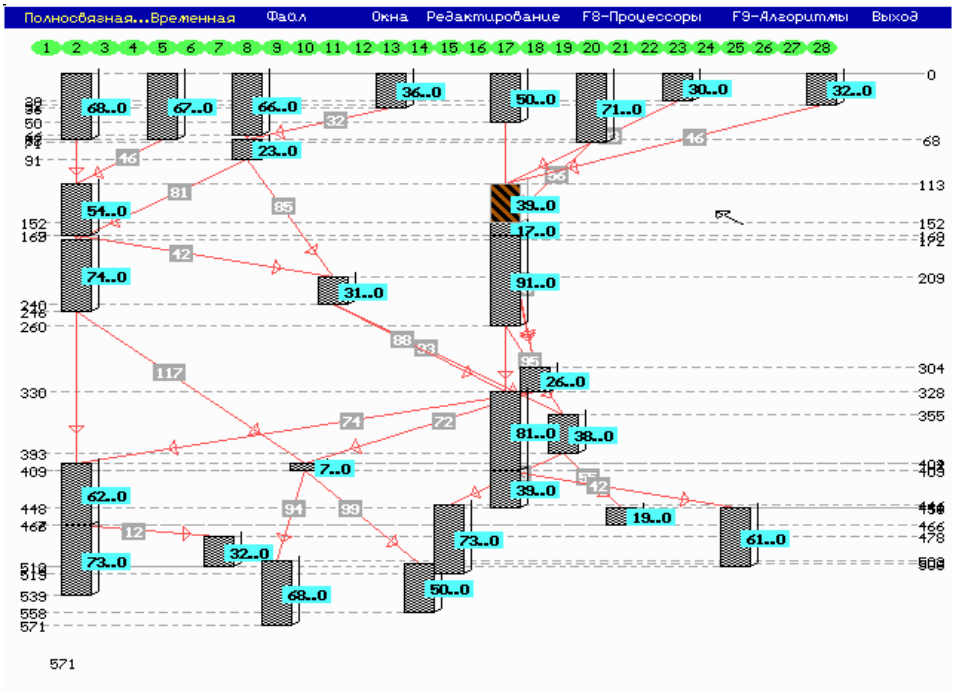


Рис. 47. Кластеризація вершини 12 з вагою 39 тактів.

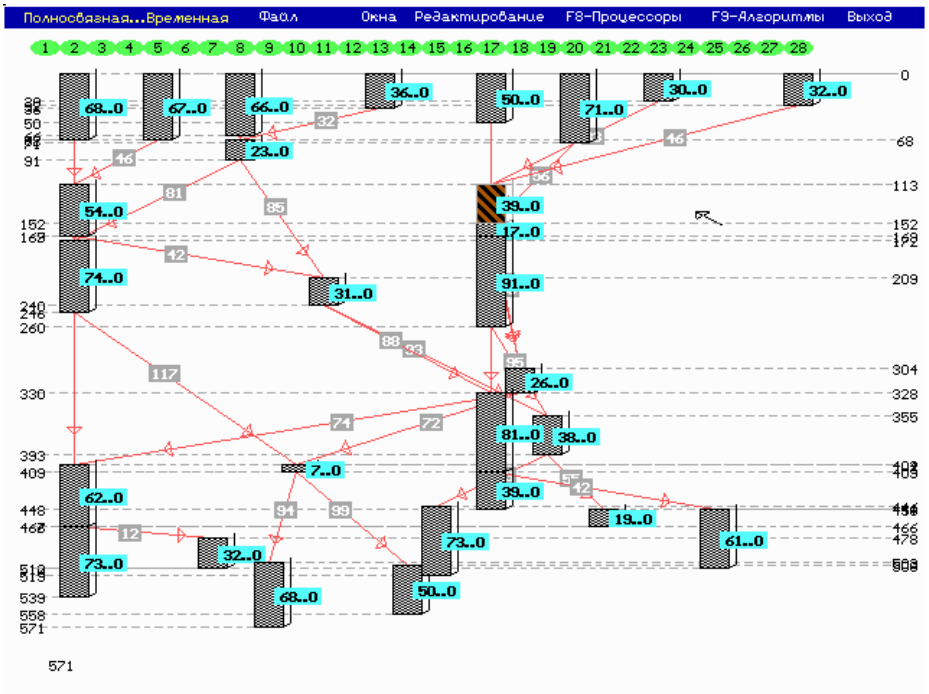


Рис. 49. Кластеризація вершини 7 з вагою 30 тактів.

Для вершини X14 (випадок № 3) виконуються умови $A_1 \subset Cpc_1$,$B_1 \subset Cpc_2$ *отримуємо такі розрахункові значення d1, d2:*

d1 (X9, X13) = 5

d2 (X16, X20) = 93

TX14 = 31

$d_2>0$ &$T_b(succ(X_{20}))$-$T'_b(succ(X_{20}))>0 \Rightarrow X_{14} \rightarrow Cpc_2$

Виконавши ці дії для вершини X15 отримаємо що $X_{15} \rightarrow Cpc_1$.

Кластеризація вершин і 14 і 15 показана на рисунку 49-50.

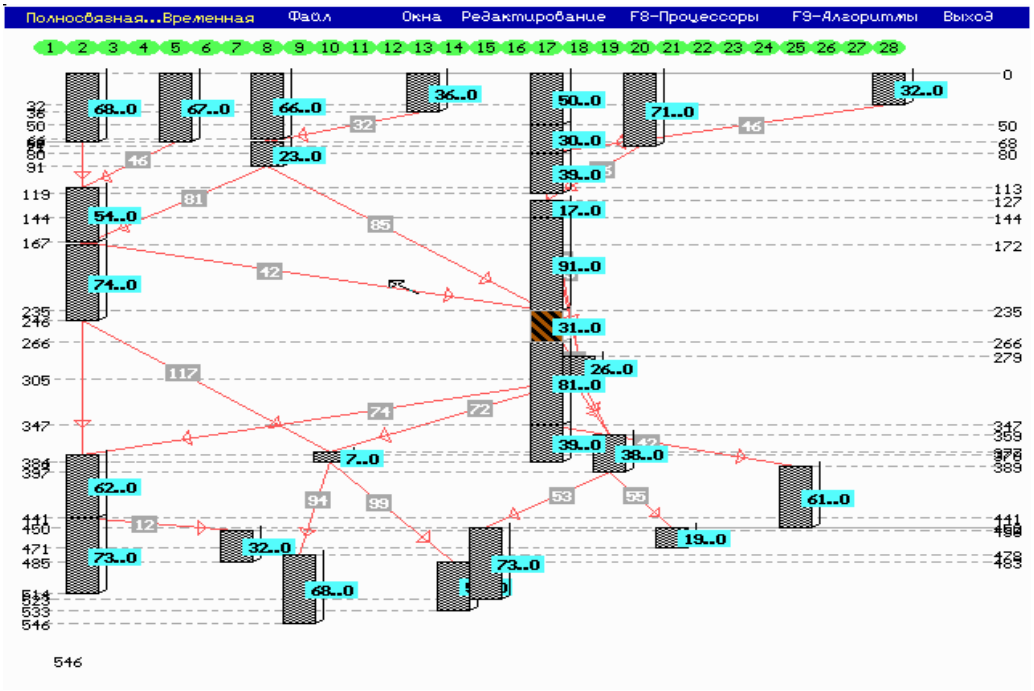


Рис. 49. Результат кластеризації вершини 14 з вагою 31 тактів.

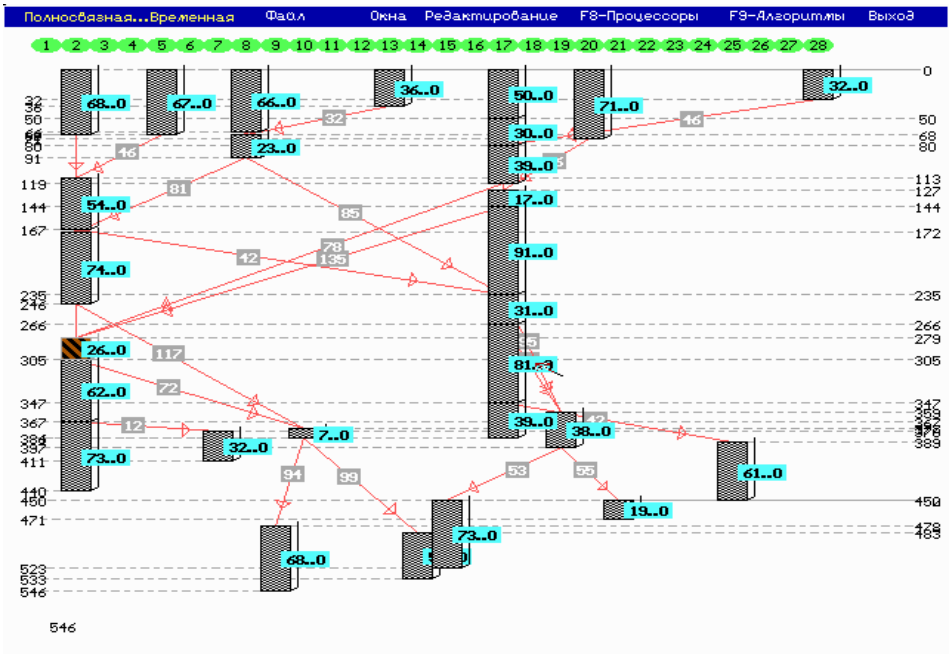


Рис.50. Результат кластеризації вершини 15 з вагою 26 тактів.

В результаті виконаних дій Ccp1 і Ccp2 оптимізовані і залишається виконати спробу кластеризації решти вершин, які не впливають на величину Ccp, але впливають на загальний час розв'язання задачі. Виконаємо проміжні обчислення.

$|T_b(X_{15})-T_f(X_{13})|=33>T_{succ(X13)}$ è $T_b(succ(X_{15}))-T'_b(succ(X_{15}) \geq 0 \Rightarrow X_{18} \rightarrow Cpc_1$.(Рис.51.)

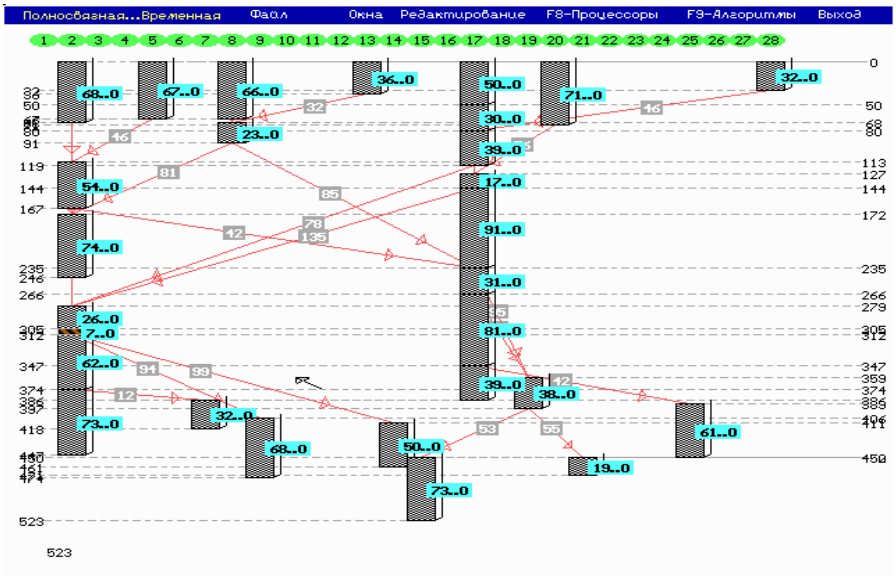


Рис. 51. Результат кластеризації вершини 18 з вагою 7 тактів.

Результат аналізу показує, для вершини X19 немає місця на процесорі, де завантажений Cpc2 але можливо розмістити max (Tb (succ (X19)) Таким чином приходимо до висновку можливості завантаження $X_{25} \rightarrow X_{19}$. (Рис. 52)

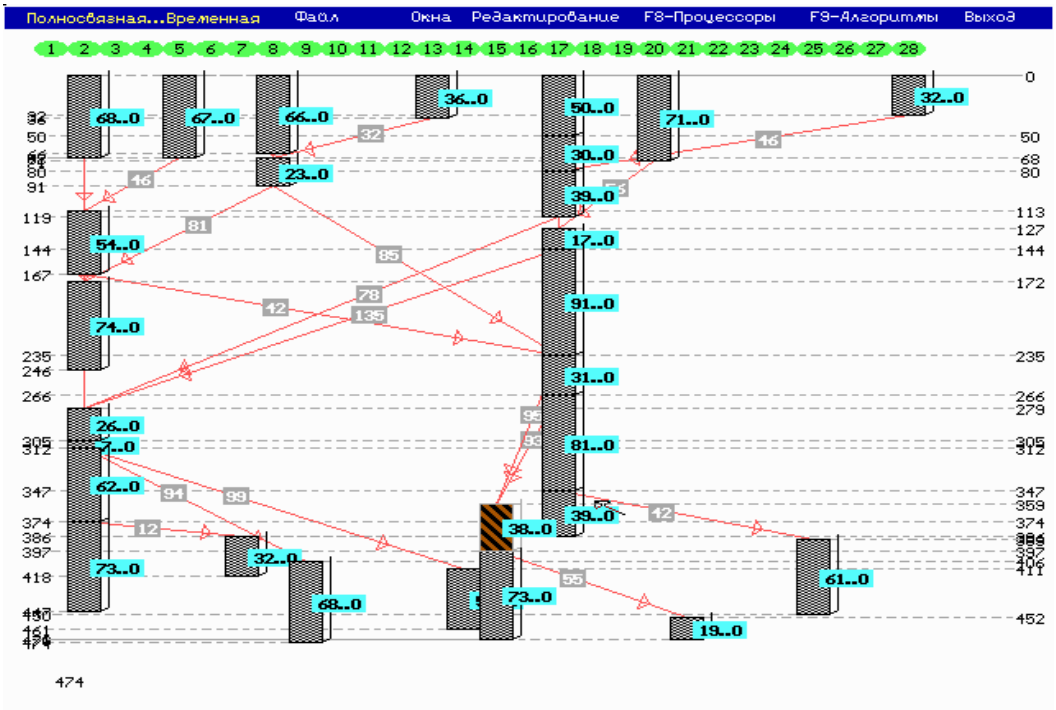


Рис. 52. Результат кластеризації вершини 19 з вагою 38 тактів.

В результаті подальшого аналізу отримуємо, що жодна з вершин графа не може бути кластерізованою з іншою вершиною зі зменшенням часу завантаження. Тобто отримано мінімальний час і потрібно мінімізувати число процесорів. На рис. № 53-54 показано виконання цих дій.

У таблиці 1 наведені результати порівняння розв'язання задачі FFT п'ятьма різними алгоритмами. Результат порівняння показує, що запропонований алгоритм дає мінімальний час рішення при однаковому числі процесорів.

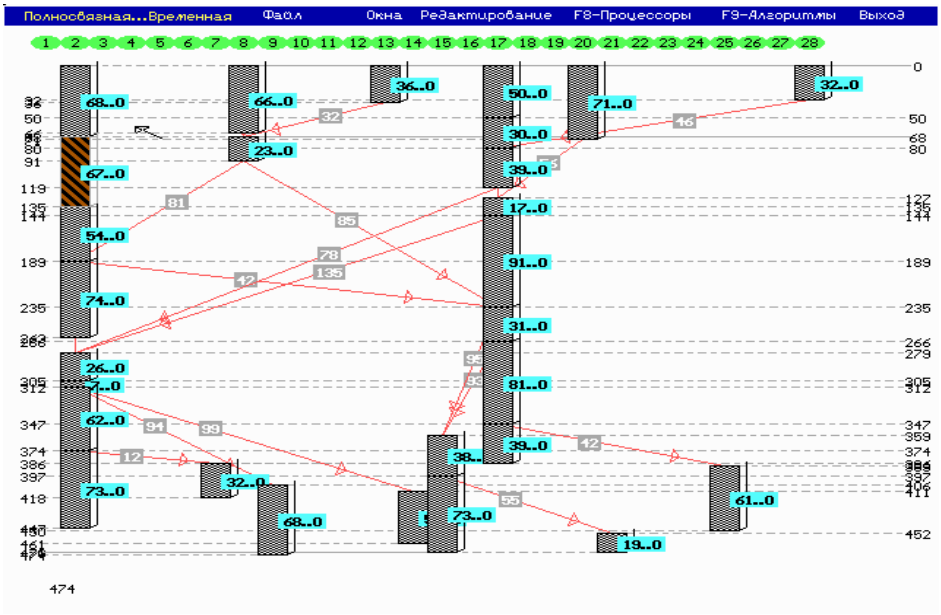


Рис. 53.

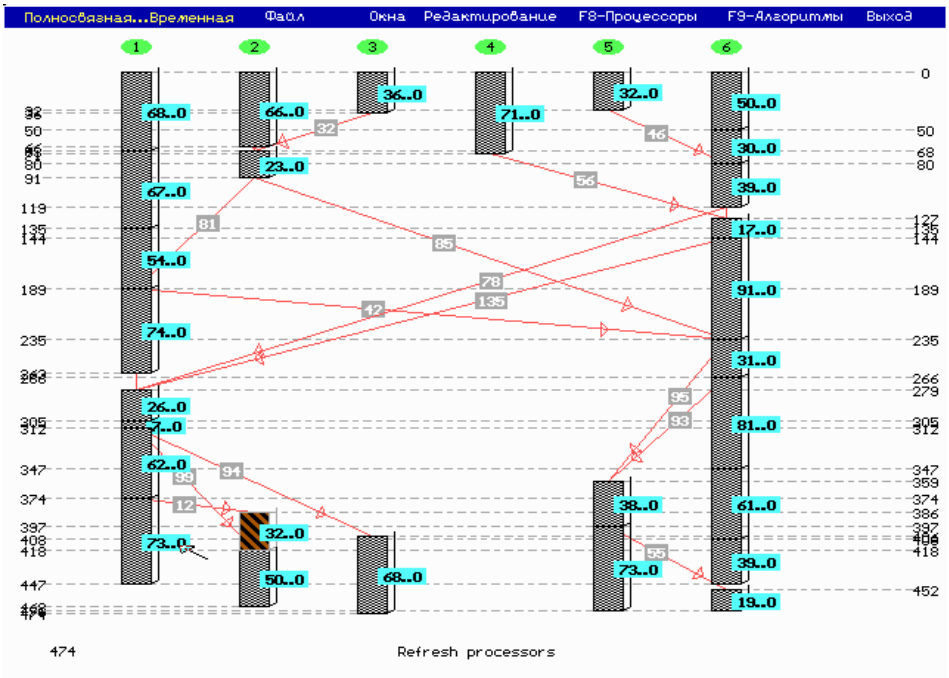


Рис. 54.

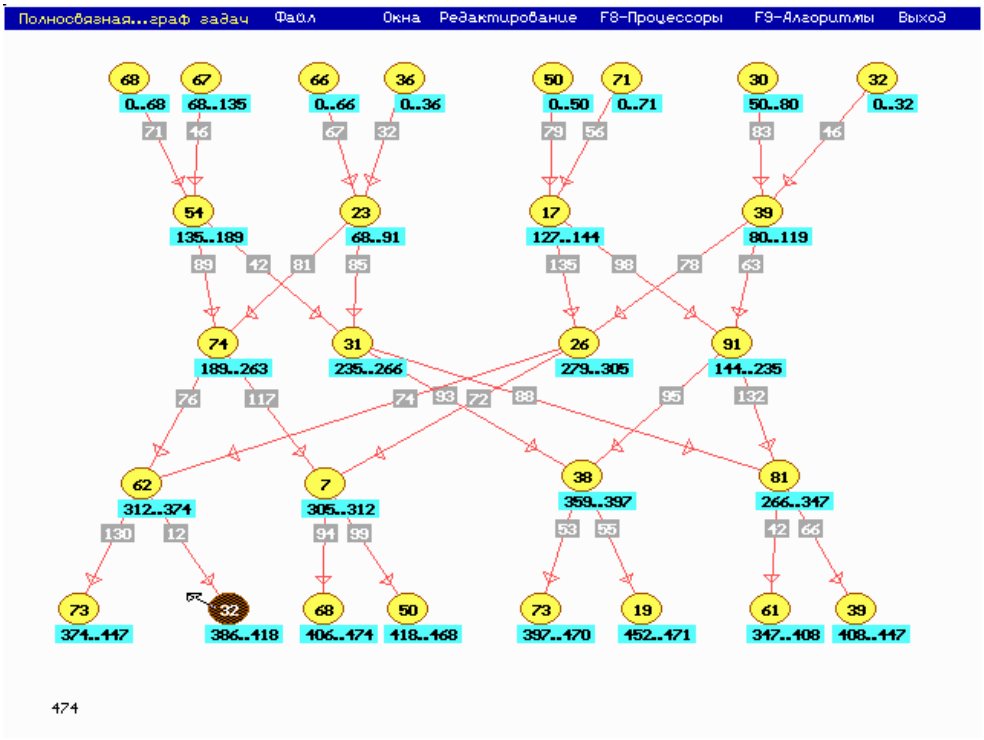


Рис. 55.Рішення задачі планування для алгоритму рівняння Лапласа

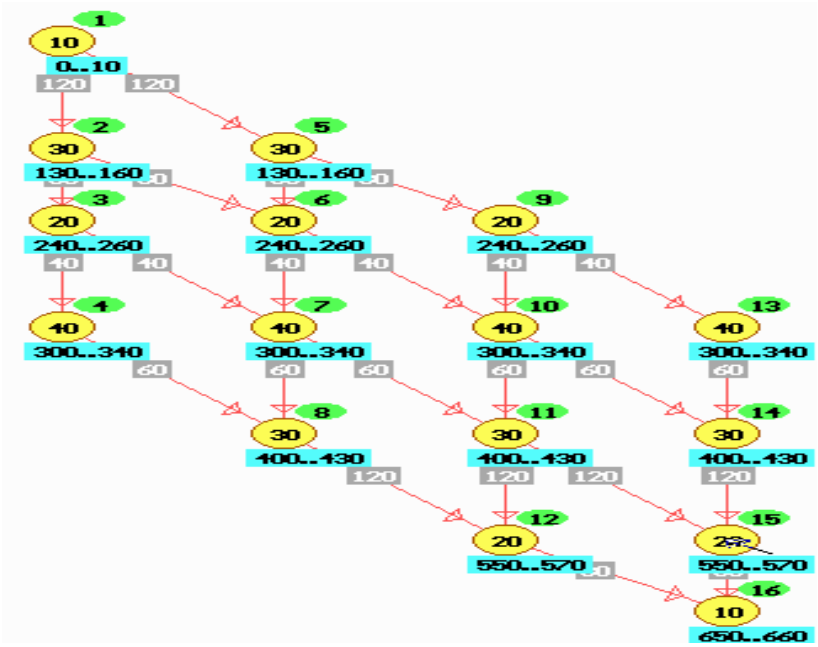


Рис. 56. Вихідний граф рівняння Лапласа.

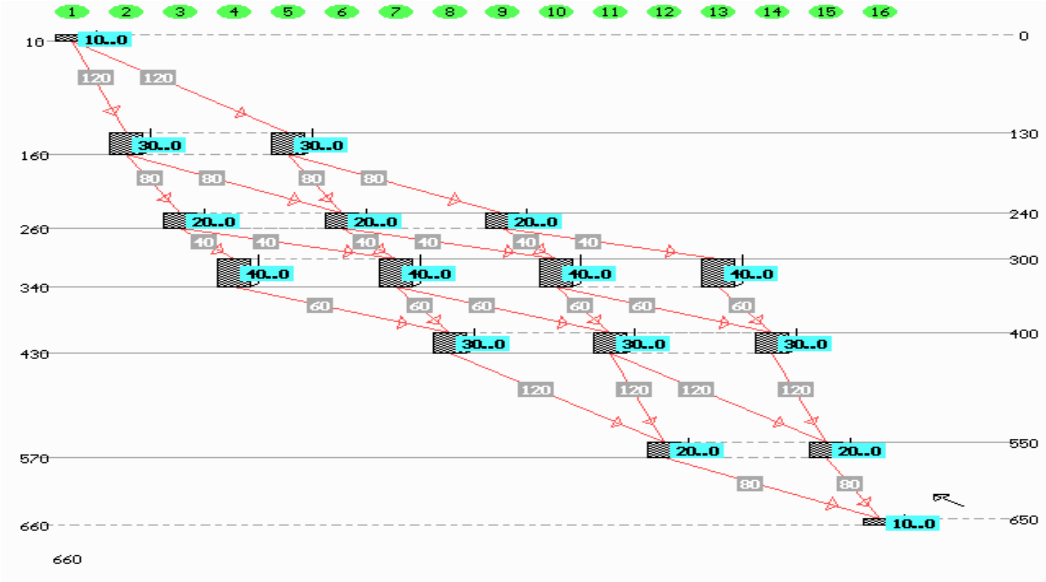


Рис. 57. Базове рішення при відсутності обмежень на число процесорів

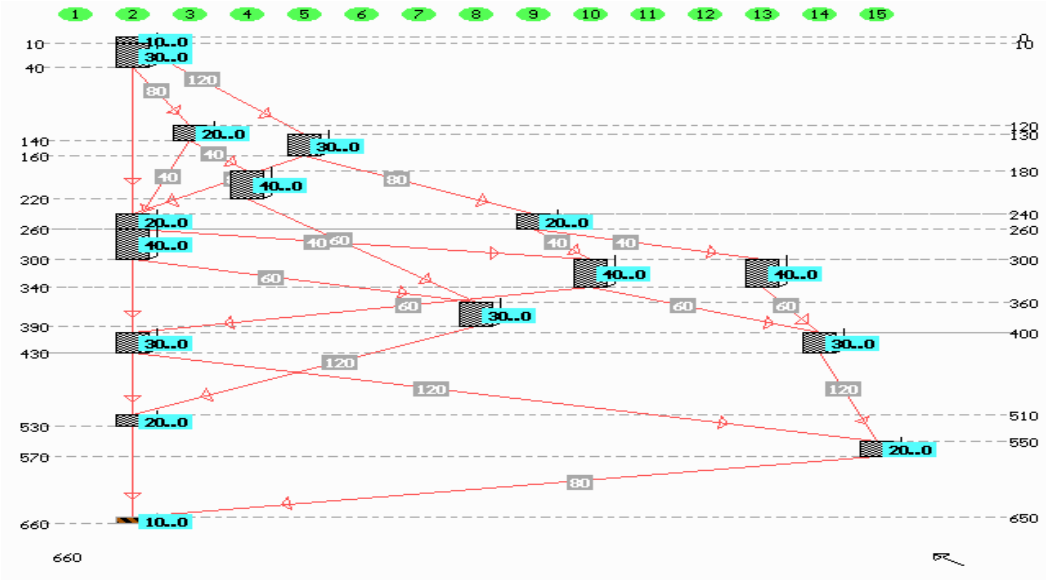


Рис. 58. Результат визначення критичного шляху.

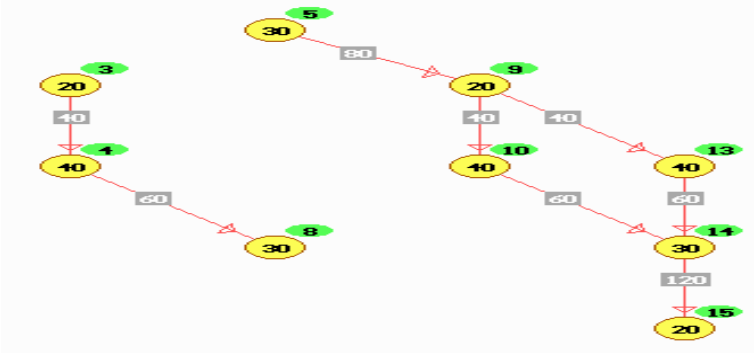


Рис. 59. Результат редукції графу і новий суграф після видалення критичного шляху з початкового графу G.

| Tf (x2)-Tb (x5) | = 200 ≠ 0 - означає, що має місце затримка і необхідний додатковий аналіз.

pred(x_5) = {x_2,x_3}, $x_2 \in$Cpc $\Rightarrow$ $T_f(x_3) + T_c(x_3 , x_5) = 240 = T_b(x_5)$- означає, що вершини x3 є причиною затримки і отже необхідна її кластеризація з вершинами на процесор, де завантажений Ccp. (Рис. 60.)

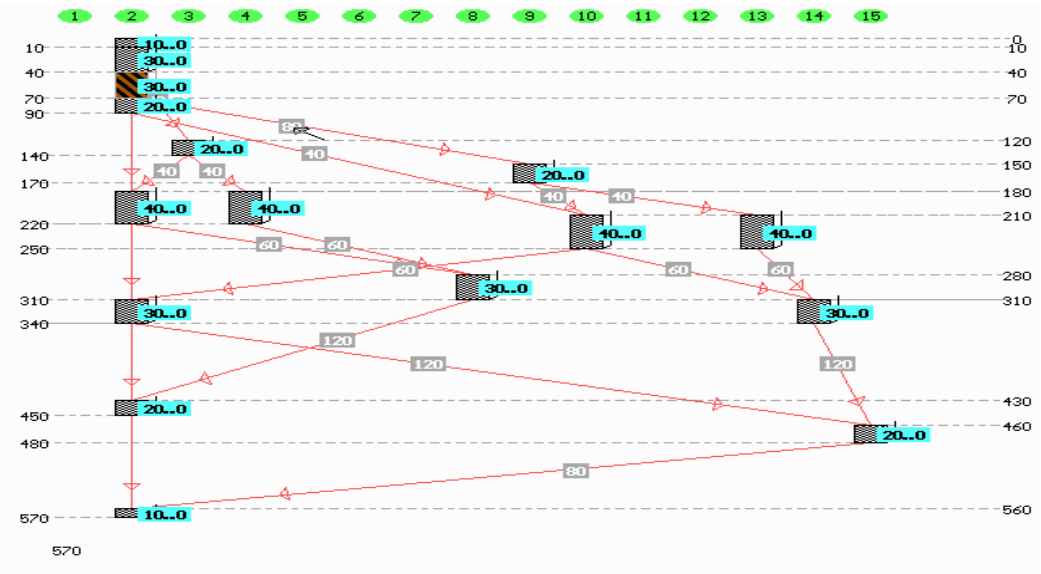


Рис. 60. План розв'язку після кластеризації вершини x3 з вершинами x2 і x5

$|T_f(x_5) - T_b(x_8)|=90\neq0$

pred(x_8)={x_4,x_5}

$T_f(x_4)+T_c(x_4,x_8)=180=T_b(x_8)$ – означає, що вершини x4 є причиною затримки і можлива її кластеризація з вершинами на процесор, де завантажені Ccp. (Рис.61.)

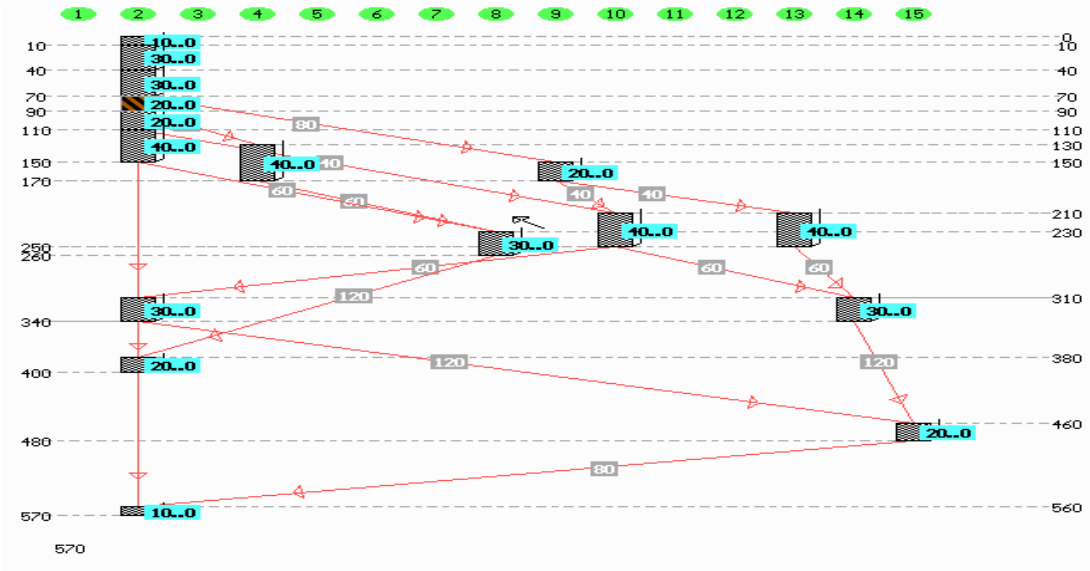


Рис. 61. Результат кластеризації вершини x4.

$|T_f(x_8)-T_b(x_{12})|=160\neq0$

pred(x_{12})={x_8,x_9}

$T_f(x_9)+T_c(x_9,x_{12})=310=T_b(x_{12})$- означає, що вершина x9 є причиною затримки і можлива її кластеризація на процесор, де завантажений Ccp. (Рис. 62.)

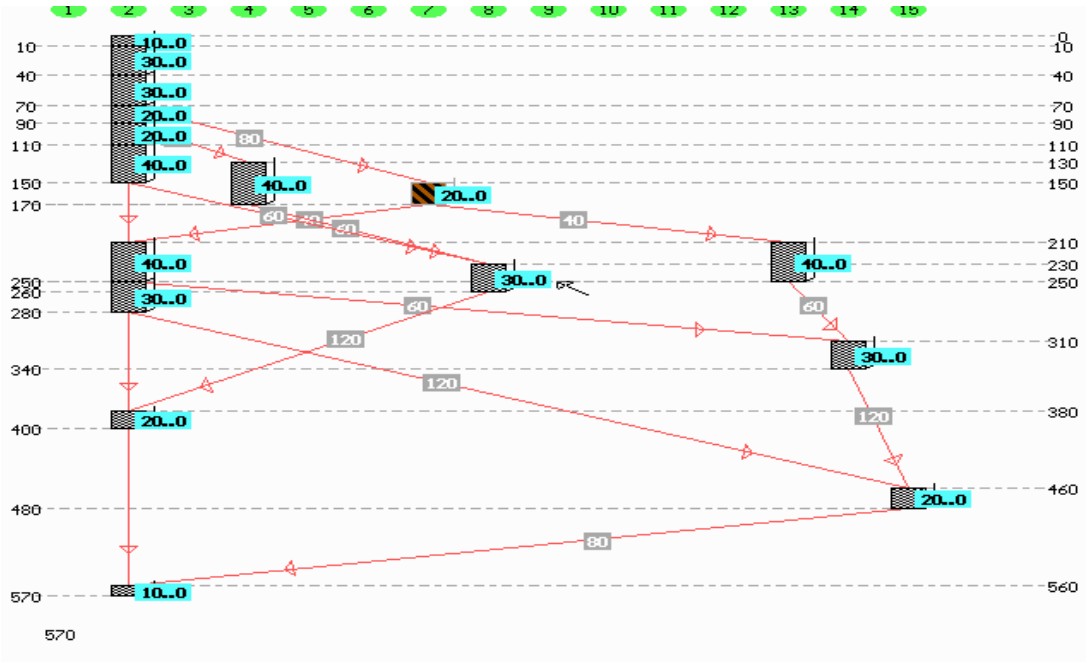


Рис. 62.Результат кластеризації вершини x9.

$|T_f(x_3)-T_b(x_9)|=140\neq 0$

$pred(x_9)=\{x_5,x_6\}$

$T_f(x_6)+T_c(x_6,x_9)=210=T_b(x_9)$- означає, що вершина x6 є причиною затримки і можлива її кластеризація на процесор, де завантажений Ccp. (Рис.63.)

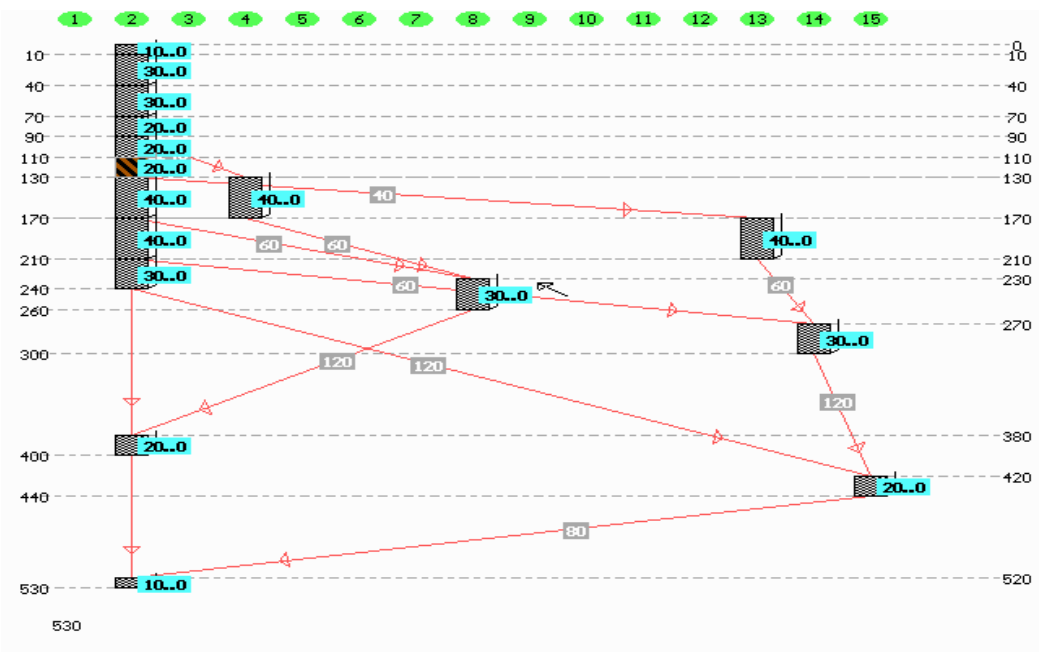


Рис. 63. Результат кластеризації вершини x6

$|T_f(x_8)-T_b(x_{14})|=210\neq 0$

$pred(x_{14})=\{x_{11},x_{12}\}$

$T_f(x_{11})+T_c(x_{11},x_{14})=380=T_b(x_{14})$- , що вершина x11 є причиною затримки і можлива її кластеризація на процесор, де завантажений Ccp. (Рис.64.)

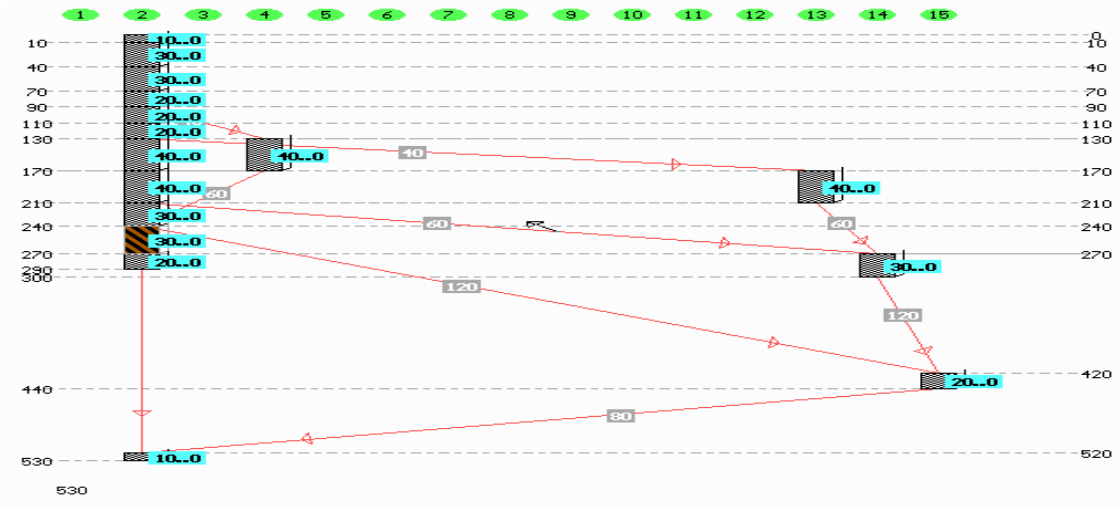


Рис. 64. Результат кластеризації вершини x11

$|T_f(x_{14})-T_b(x_{16})|=230\neq 0$

$pred(x_{16})=\{x_{14},x_{15}\}$

$T_f(x_{15})+T_c(x_{15},x_{16})=520=T_b(x_{16})$

Це означає, що вершина x15 є причиною затримки і можлива її кластеризація на процесор, де завантажений Ccp (Рис.65.)

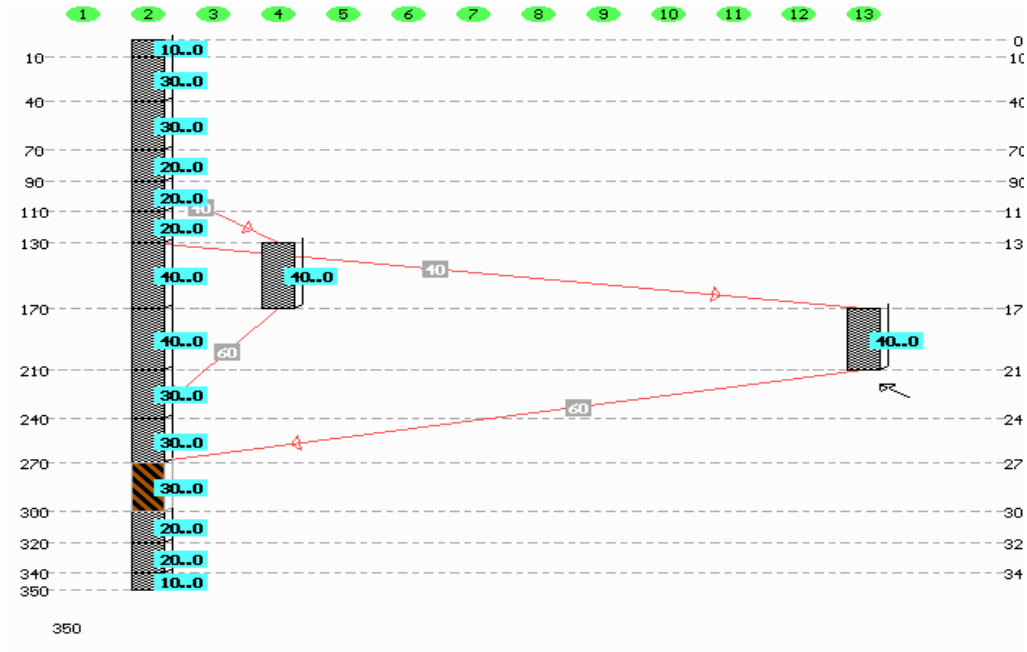


Рис. 65. Результат кластеризації вершини x15

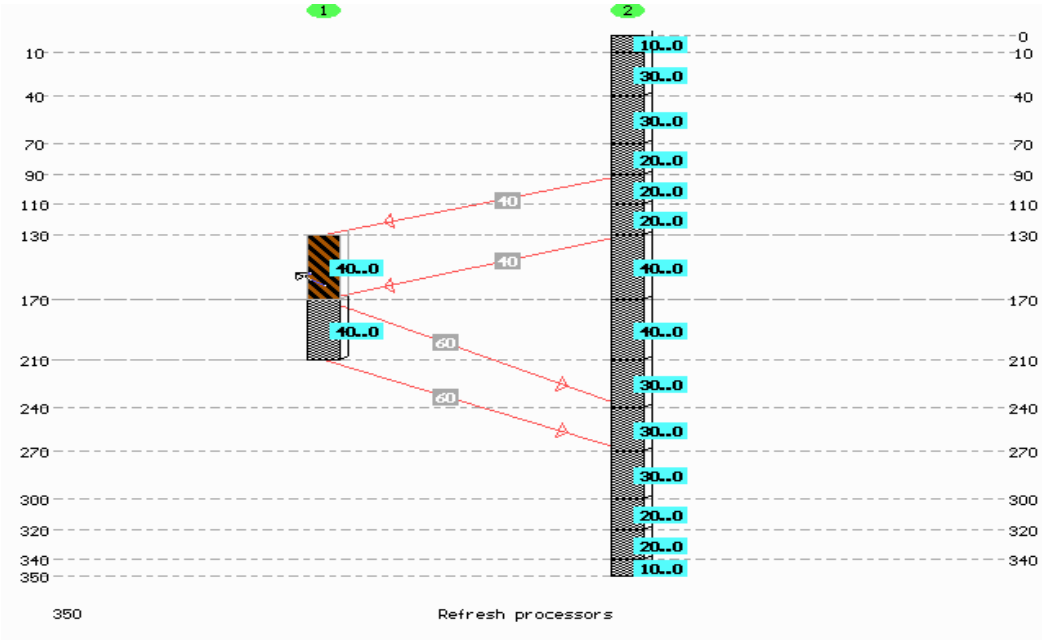


Рис. 66. Остаточний план завантаження алгоритму задачі Лапласа.

Приклад складання плану завантаження розв'язання задачі виключення Гауса

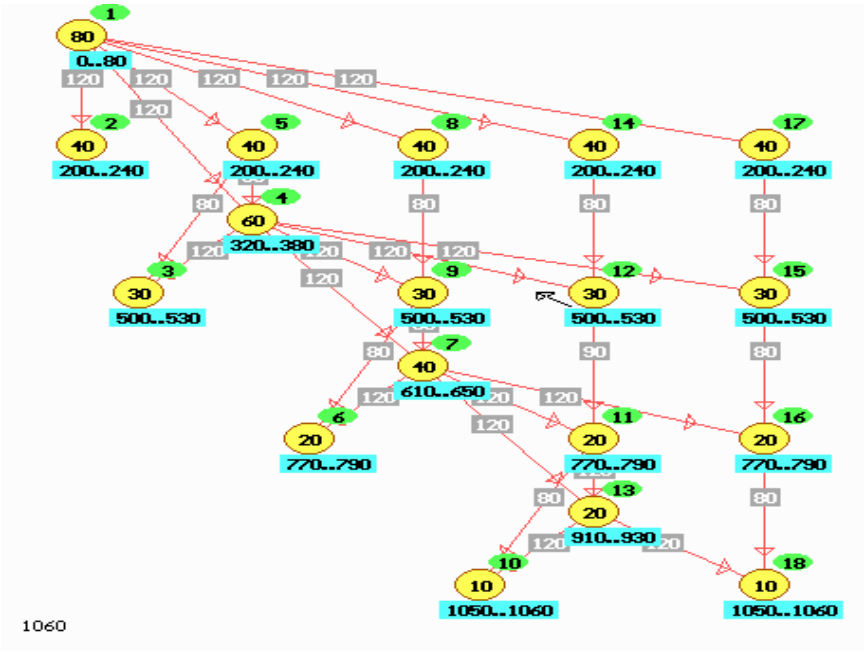


Рис. 67. Вихідний граф розв'язання задачі виключення Гауса.

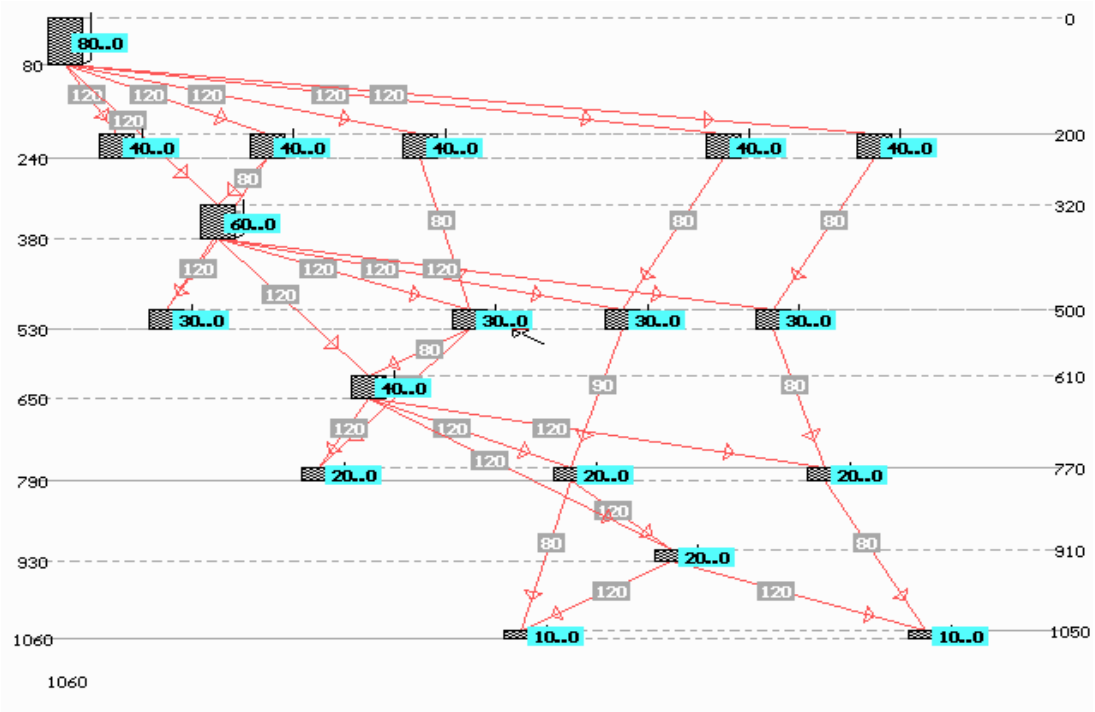


Рис. 68. Базове рішення при відсутності обмежень на кількість процесорів

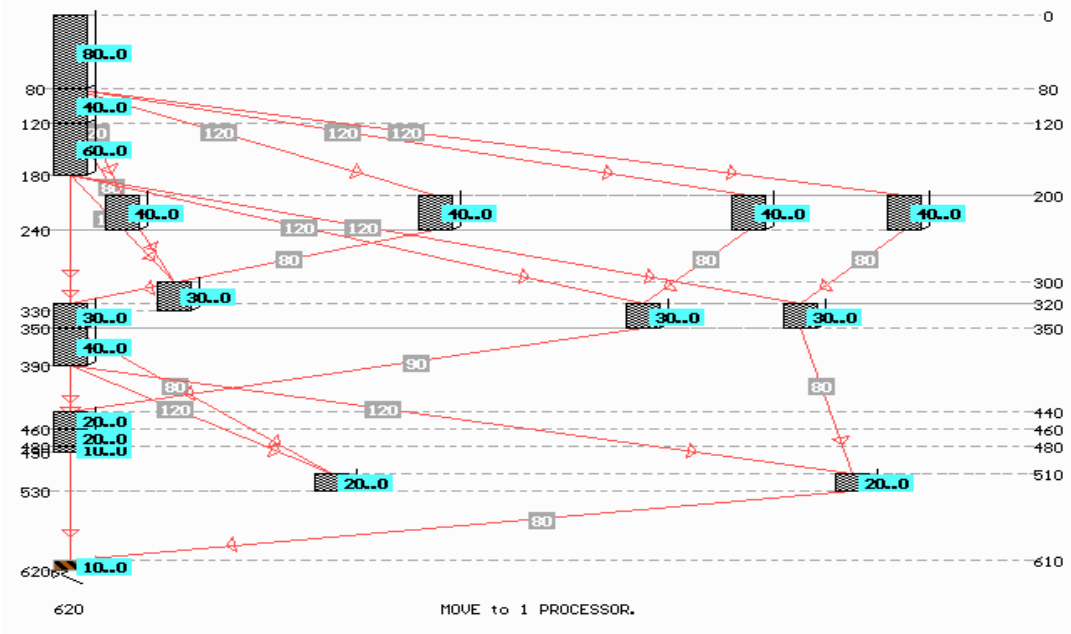


Рис. 69. Результат виконання першого кроку і завантаження критичного шляху.

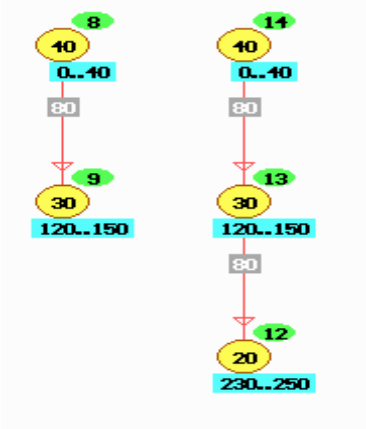


Рис 70. Результат редукції вершин СРС.

$|T_f(x_7)-T_b(x_9)|=140\neq 0$

$pred(x_9)=\{x_4,x_7\}$

$T_f(x_4)+T_c(x_4,x_9)=320=T_b(x_9)$- означає, що вершина x4 є причиною затримки і можлива її кластеризація нав процесор, де завантажений Ccp (Рис. 71.)

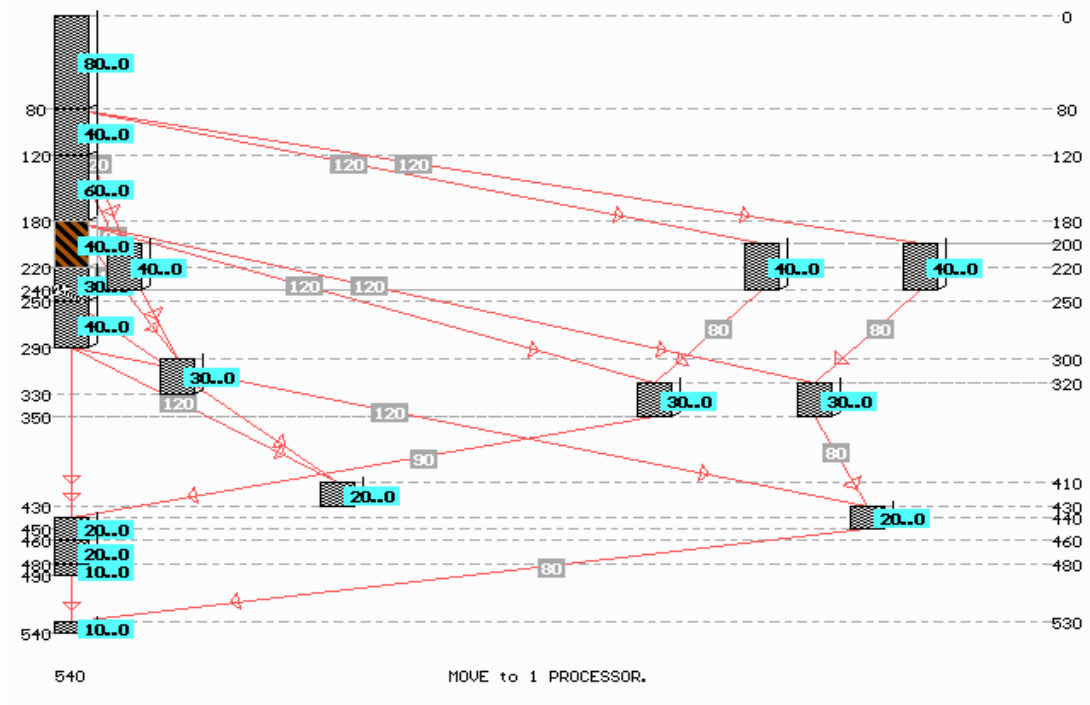


Рис 71. Результат кластеризації вершини x4

$|T_f(x_{12})-T_b(x_{14})|=140\neq 0$

$pred(x_{14})=\{x_{10},x_{12}\}$

$T_f(x_{10})+T_c(x_{10},x_{14})=430=T_b(x_{14})$- означає, що вершина x10 є причиною затримки і можлива її кластеризація на процесор, де завантажено Ccp. (Рис. 72.)

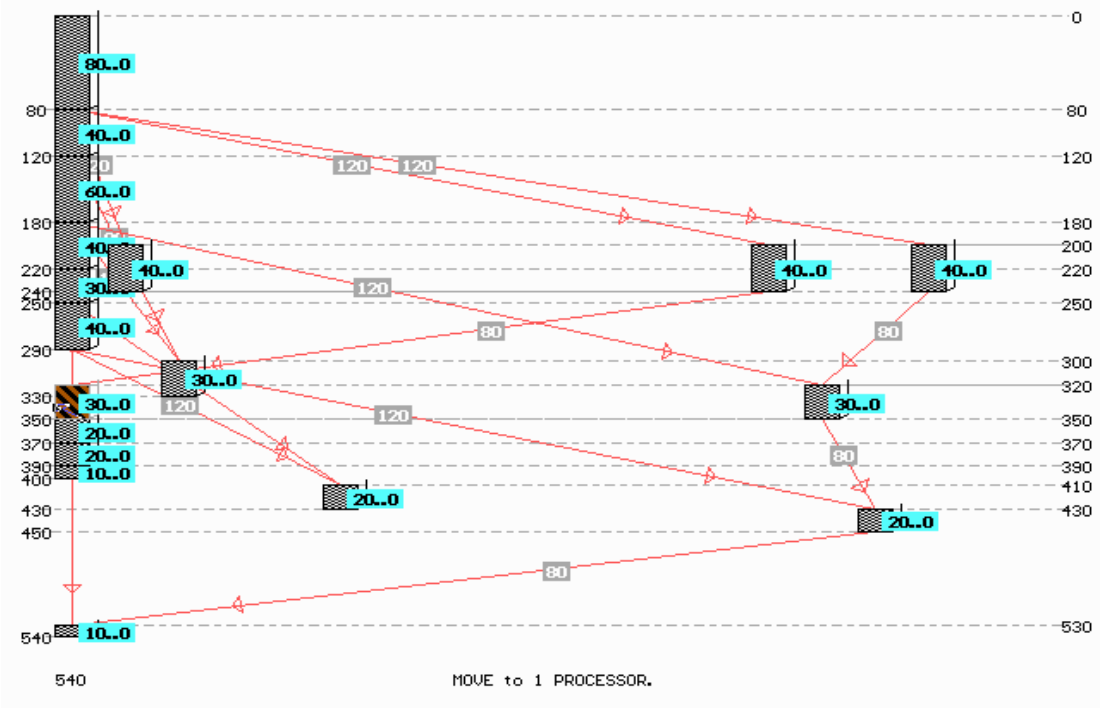


Рис 72. Результат кластеризації вершини x10

$|T_f(x_{16})-T_b(x_{18})|=140\neq 0$

$pred(x_{18})=\{x_{15},x_{16}\}$

$T_f(x_{15})+T_c(x_{15},x_{18})=530=T_b(x_{18})$- означає, що вершина x15 є причиною затримки і можлива її кластеризація нà процесор, де завантажений Ccp. (Рис. 73.)

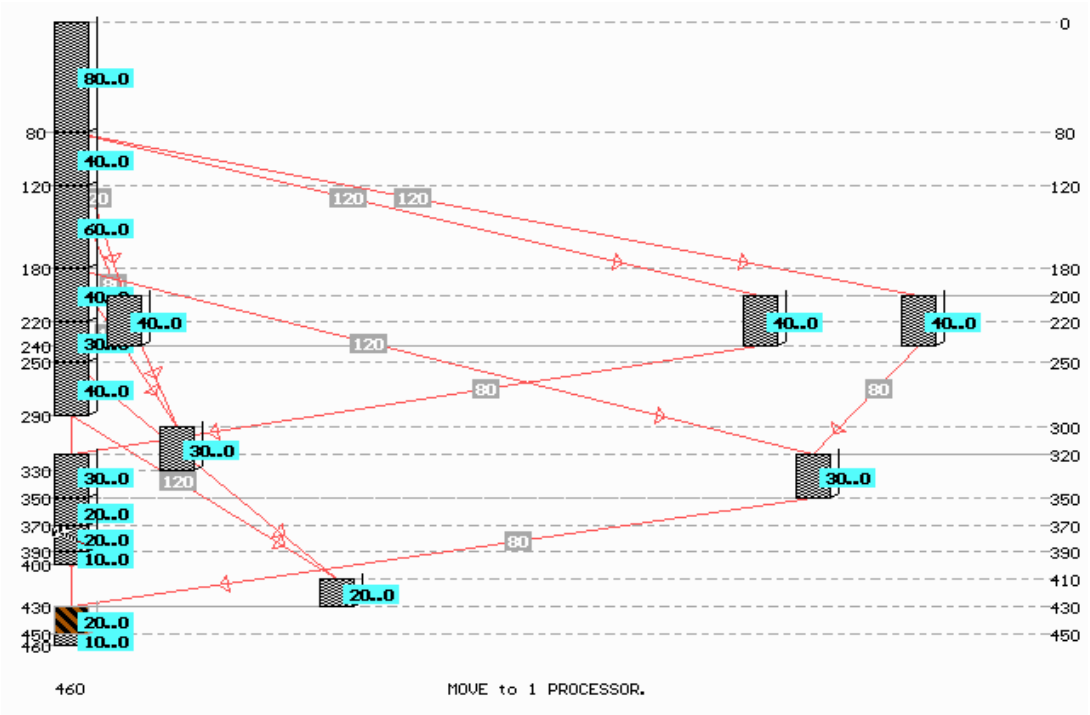


Рис 73. Результат кластеризації вершини x15

$|T_f(x_{17})-T_b(x_{15})|=30\neq 0$

$pred(x_{15})=\{x_{11},x_{12}\}$

$T_f(x_{11})+T_c(x_{11},x_{15})=430=T_b(x_{15})$

$T_{x11}=30$ - означає, що кластеризація вершини x11 на процесор, де завантажений Ccp не зменшує час рішення і її кластеризація не має сенсу.

$pred(x_{11})=\{ x_6,x_7\}$

$T_f(x_6)+T_c(x_6,x_{11})=320= T_b(x_{11})$

$T_f(x_7)+T_c(x_7,x_{11})=300 \neq T_b(x_{11})$ - означає, що кластеризація вершини x11 на процесор, де завантажена вершина x6 зменшує час планування завдання. (Рис. 74.)

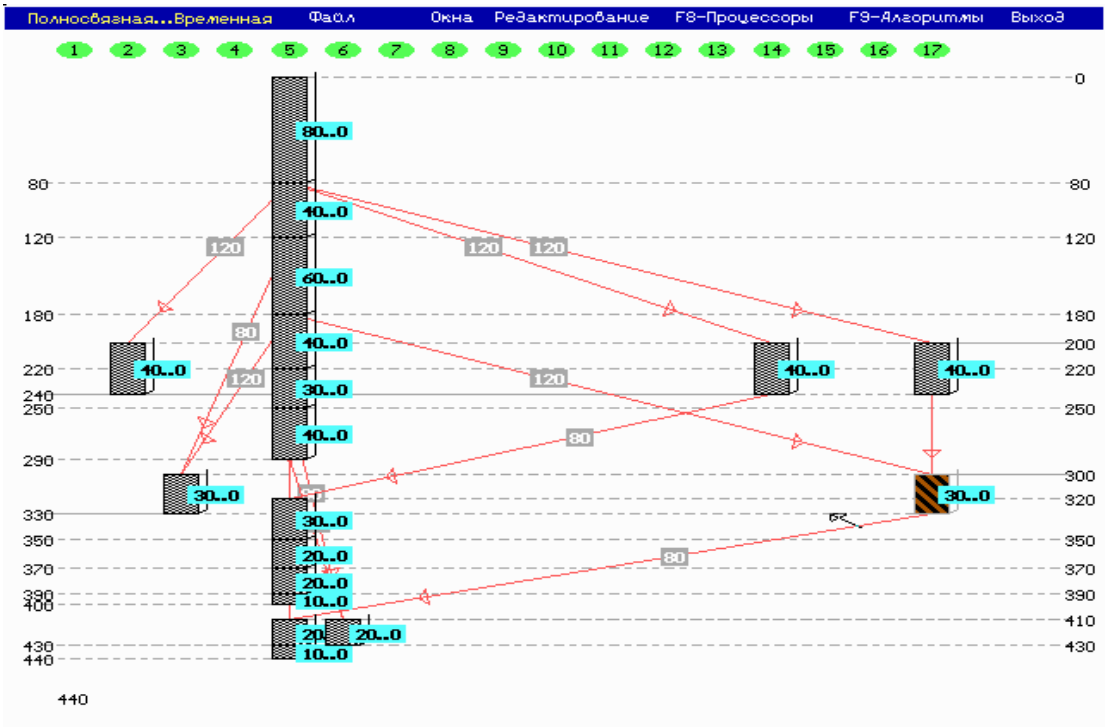


Рис 74. Результат кластеризації вершини x11

$|T_f(x_{12})-T_b(x_{10})|=30\neq 0$

$|T_f(x_{17})-T_b(x_{15})|=10\neq 0$

вершини x10, x12, x15, x17 знаходяться на CPC. Різниця між сумою ваг вершин і часом планування дорівнює 40 тактів, але pred (x10) = {x5, x7} і Tx5 = 40 тактів. Таким чином її кластеризація не збільшує час розв'язання задачі і вершина x5 може бути скластерізована на процесор, де завантажений CPC. (Рис. 75.)

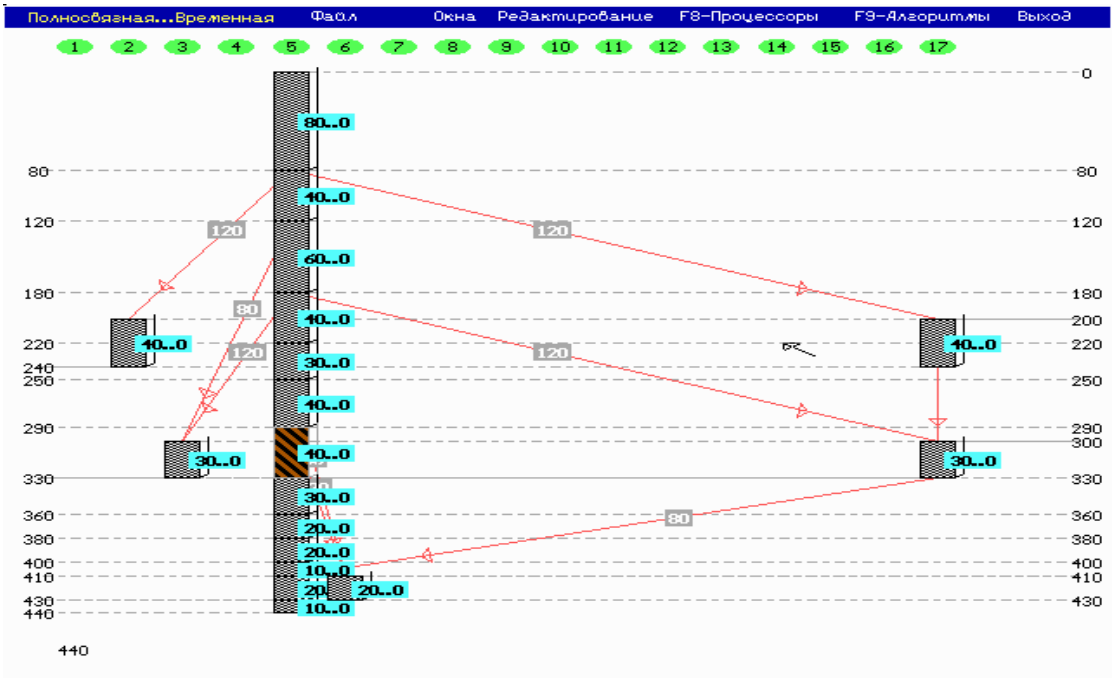


Рис 75. Результат кластеризації вершини x5

Кластеризація всіх інших вершин тільки зменшує кількість процесорів для вирішення цього завдання без збільшення часу виконання.

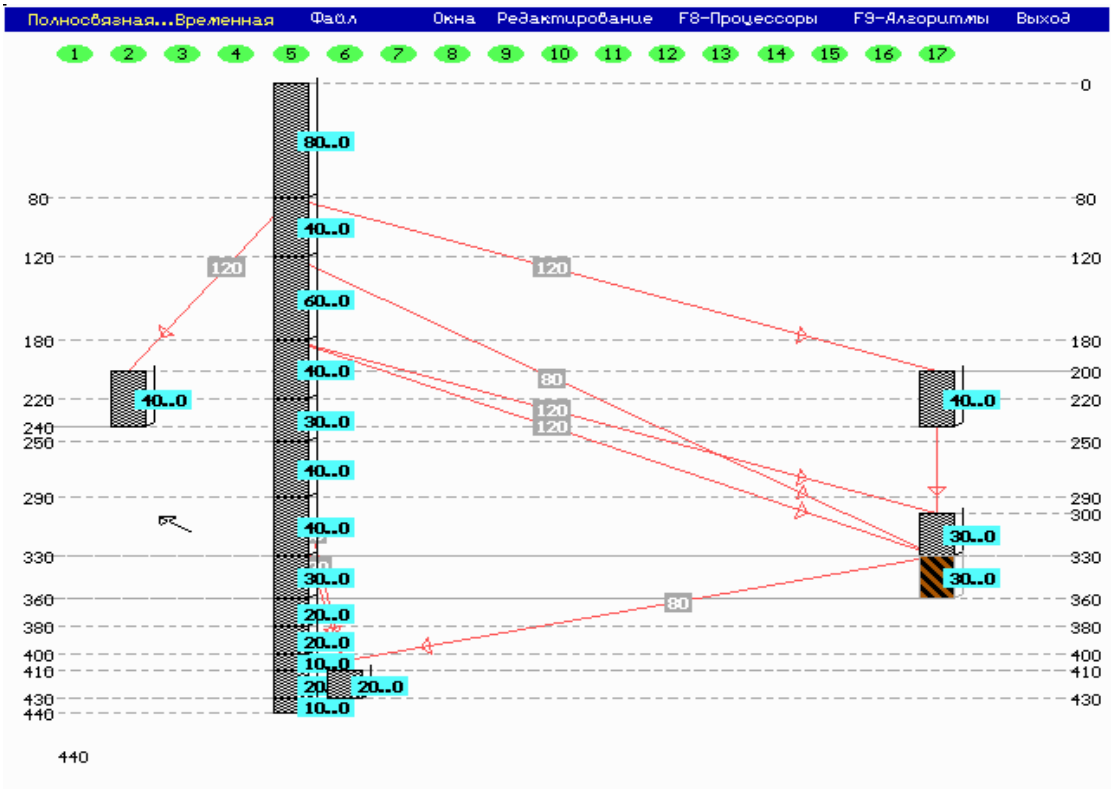


Рис 76. Результат кластеризації вершини x8

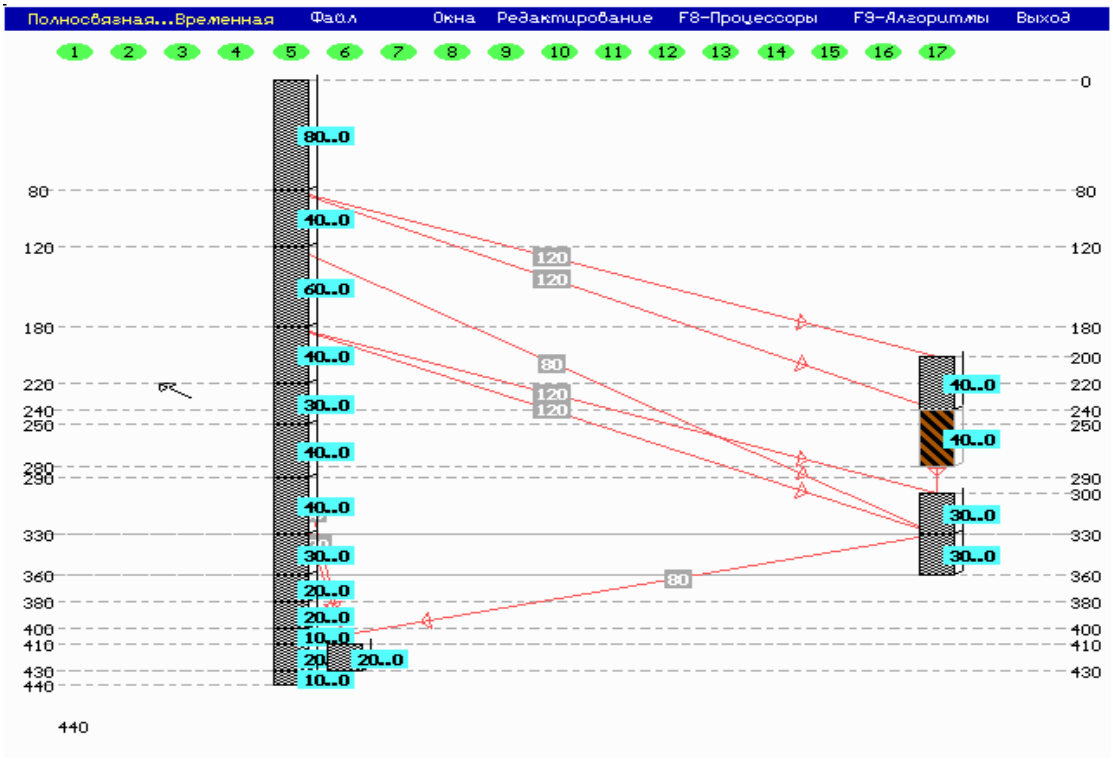


Рис 77. Результат кластеризації вершини x2

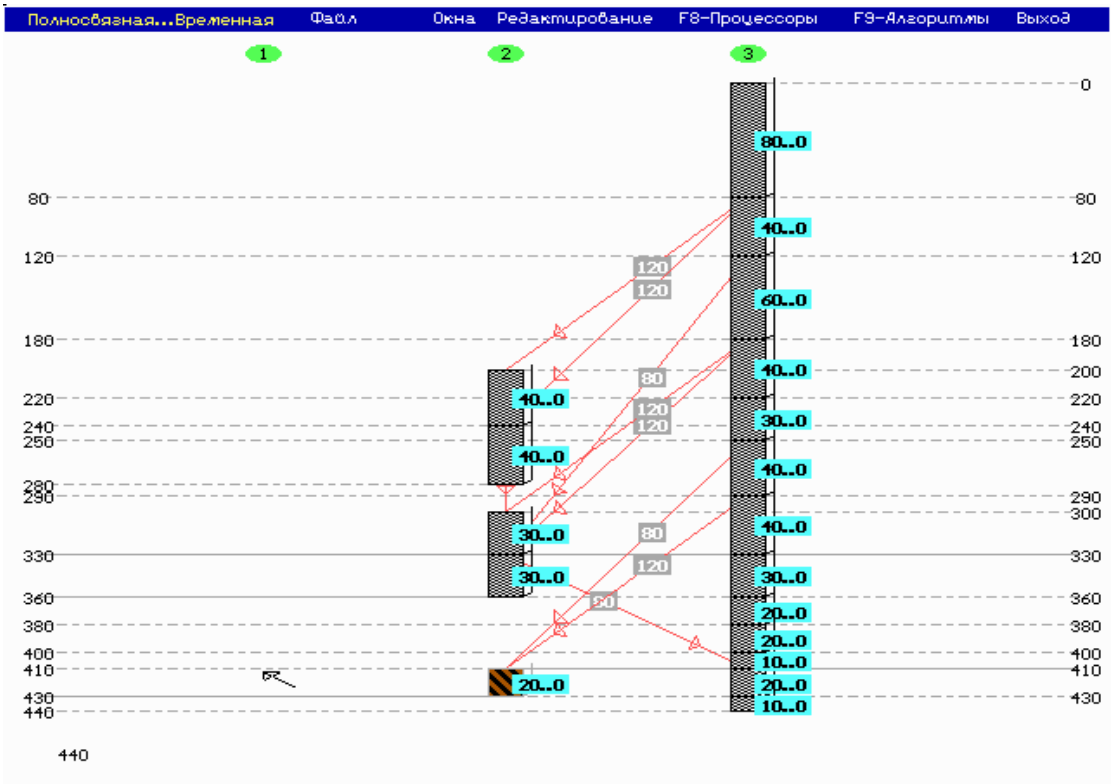


Рис 78. Результат кластеризації вершини x13.Кінцевий варіант планування

Алгоритм DCPC поділяється на три частини:

1. Алгоритм знаходження критичного шляху.

2. Алгоритм знаходження причини затримки (вершини).

3. Алгоритм оптимізації кількості процесорів.

На тимчасову складність DCPC найбільшою мірою впливає тимчасова складність пошуку критичного шляху, тому що інші частини виконують локальну оптимізацію і мають значно меншу тимчасову складність. Алгоритм знаходження затримки виконується один раз і його тимчасова складність дорівнює << O (n). Алгоритм зменшення кількості процесорів також виконується один раз і його тимчасова складність << O (n).

Таким чином тимчасова складність алгоритму DCPC визначається часової складністю алгоритму критичного шляху і дорівнює O (2 (n '+ e)) або O (n (n1) +2 e). У таблиці 5 виконаний порівняльний аналіз часової складності 8 найбільш відомих алгоритмів із запропонованим.

Табл. 5.

Алгоритм	Складність
EZ	O(e(e+V)
MCP	$O(v^2 \log v)$
MD	$O(v^3)$
EFT	$O(pv^2)$
DLS	$O(v^3 pf(p))$
DSC	O((e+v)logv)
DCP	$O(v^3)$
KIM	O(n log n)
DCPC Запропонований алгоритм	O(n(n-1)+2e)

Завдання на курсовий проект

Згідно з варіантом запрограмувати просторовий планувальник для розподілу завдань у багато процесорній системи і дослідити його роботу покроково .

Просторово - часове планування робіт в обчислювальних системах. Виконання роботи містить слідуючи пункти:

1. Попередній аналіз заданої інформації для визначення критичного шляху, необхідного числа процесорів, максимального ступеню розпаралелювання.
2. Попередній розподіл робіт в обчислювальній системі.
3. Оптимізація розкладу завантаження обчислювальних елементів по заданим критеріям оптимізації.
4. Оцінка якості отриманого розкладу.

Вхідні дані:

- Не менше як 15 вузлів в графі
- Граф орієнтованій, ациклічний
- Топологія багато процесорній системи згідно варіанту

Варіант курсової роботі береться згідно номеру залікової книжці з наступного списку:

1. Загальна шина
2. Повнозв’язна система
3. Загальна шина с маркерним доступом
4. Зірка с активним центром
5. Зірка с мостом
6. Зірка с комутатором
7. Деревовидна структура
8. Вектор
9. Ланцюг процесорів
10. Векторне кільце
11. Коло процесорів
12. Матриця 3x3
13. Загальна шина - неоднорідна
14. Повнозв’язна система - неоднорідна

15. Загальна шина с маркерним доступом - неоднорідна
16. Зірка с активним центром - неоднорідна
17. Зірка с мостом - неоднорідна
18. Зірка с комутатором - неоднорідна
19. Деревовидна структура - неоднорідна
20. Вектор - неоднорідна
21. Ланцюг процесорів - неоднорідна
22. Векторне кільце - неоднорідна
23. Коло процесорів - неоднорідна
24. Матриця 3х3 - неоднорідна
25. ТОР 3х3 - неоднорідна
26. Гиперкуб - неоднорідна
27. Token – Ring
28. FDDI

Джерела інформації

[1] Алгоритмы и программы решения задач на графах и сетях / Под. ред. М. И. Нечепуренко и др.-Новосибирск: Наука. Сиб. отд-ние, 1990.

[2] Барский А.Б., Параллельные процессы в вычислительных системах. Планирование и организация., М., Радио и связь, 1990г.

[3] Басакер З., Саати Т.Л., Конечные графы и сети, М., Наука, 1974 г.

[4] Березин Е.А., Оптимальное распределение ресурсов и элементы синтеза систем., М., Сов. радио, 1974 г.

[5] Белинский В.А., Беляев В.Г., Организация вычислительного процесса с использованием комплекса программ диспетчеризации и учета//М.,1986,№1.

[6] Берж К., Теория графов и ее применения. - М.: Изд-во иностр. лит.(ИЛ), 1962.

[7] Бронштейн О.И., Духовный И. М., Модели приоритетного обслуживания в информационно-вычислительных системах. Москва,Наука, 1976.

[8] Вальковский В.А., Распараллеливание алгоритмов и программ. Структурный подход., М., Радио и связь, 1989 г.

[9] Визинг В.Г., Алгоритм оптимального подбора интенсивностей выполнения работ. Методы и решения оптимизационных задач на графах и сетях. Тез. докл II Всесоюзного совещ., Улан-Уде, 24-26 августа 1982 г., Новосибирск: ВЦ СО АН СССР, 1982 г. с. 42-45.

[10] Воеводин В.В., Математические модели и методы в параллельных процессах, М., Наука, 1986 г.

[11] Гэри М.Р., Джонсон Д.С., Вычислительные машины и труднорешаемые задачи .- М., Мир, 1982- 416 с.

[12] Головкин Б.А., Параллельные вычислительные системы, М. Наука, 1983г.

[13] Дроздов Е.А., Пятибратов А.П., Основы построения и функцинирования вычислительных систем, М., Энергия,1973.

[14] Диниц Е. А., Алгоритм решения задач о максимальном потоке в сети со степенной оценкой.- Докл. АН СССР.-1970.-Т.194, N 4.- С. 754- 757.

[15] Зыков А.А., Основы теории графов, М. Наука, 1987 г., 380.

[16] Каган Б.М., Электронные вычислительные машины и системы, Учебное пособие для вузов, 3-е изд., М. Энергоатомиздат, 1991г.

[17] Каляев А.В., Организация многопроцессорных систем на принципах потока данных и программирования архитектуры // Высокопроизводительные вычислительные системы, Тез. докл 2-го Всесоюз. совещания, Батуми, 1984г.

[18] Казаринов Л.С., Мерензон М.Н., Планирование параллельных вычислений в управляющих МВС при дефиците ресурсов. Управляющие системы и машины., №2., 1988 г.

[19] Карзанов А. В., Нахождение максимального потока в сети методом предпотоков.- ДАН СССР, 1974. т. 215, N 1. с.49-52.

[20] Клейнрок Л., Вычислительные системы с очередями, М. Мир, 1978г.

[21] Кристофидес Н., Теория графов. Алгоритмический подход. -М.: Мир, 1978.

[22] Ларионов А.М., Майоров С.А., Новиков Г.И., Вычислительные комплексы, системы и сети, Л. , Энергоавтомиздат, 1987.

[23] Липски В., Комбинаторика для программистов. -М.:Мир,1988.

[24] Липаев В.В., Распределение ресурсов в вычислительных системах, М. Статистика, 1979 г.

[25] Майника Э., Алгоритмы оптимизации на сетях и графах. -М.: Мир, 1981.

[26] Миренков Н.Н., Параллельное программирование для многомодульных вычислительных систем. М., Радио и связь, 1989 г.

[27] Мова В.В., Пономаренко Л.А., Калиновский А.М., Организация Приоритетного Обслуживания в АСУ. Киев, Техника,1977.

[28] Мультипроцессорные вычислителные системы/Под ред. Проф. Я.А. Хетагуравова. М., Энергия.

[29] Оре О., Теория графов. М.:Наука,1968. (1980)

[30] Поспелов Д.А., Введение в теорию вычислительных систем., М. Сов. радио, 1972 г.

[31] Прангишвили И.В., Виленкин С.Я., Параллельные вычислительные системы с общим управлением, М., Энергоатомиэдат, 1983 г.

[32] Самофалов К.Г., Луцкий Г.М., Симоненко В.П., Метод предварительного анализа исходной информации и выработки стратегии решения задачи назначения в распределенной вычислительной среде, Ж., "Электронное моделирование", № 4, 1995 г.

[33] Свами М., Тхуласирами К. Графы, сети и алгоритмы. -М.: Мир, 1984.

[34] Симоненко В.П. Организация вычислительных процессов в ЭВМ, комплексах, сетях и системах, Киев, "Век +", 1997 г., 304 стр.

[35] Симоненко В. П., Самофалов К.Г., Автоматизация составления расписания занятий в ВУЗе, Киев,МВ и ССО УССР ,1973 г.

[36] Симоненко В.П., Теоретические основы быстрого поиска максимального паросочетания, Деп. в ВИНИТИ 07.02.95, № 276-Ук95 Киев,1995 г.

[37] Симоненко В.П., С.Лахиани, Симоненко А.В. Алгоритм распределения работ в параллельной вычислительной системе, Деп. в ВИНИТИ 07.02.95, № 278-Ук95 Киев,1995 г.

[38] Симоненко В.П., Разработка и исследование алгоритмов, программ и технических средств для системы планирования учебного процесса ВУЗа, Автореф. на соискание ученой степени к.т.н., Киев, 1972.

[39] Симоненко В.П., Обоснование выбора метода динамической диспетчеризации работ в распределенных ВС, Ж. "Электронное моделирование", № 4, 1997 г., 28 стр.

[40] Самофалов К.Г., Симоненко В.П., Пространственно-временная модель планирования работ в параллельной вычислительной системе, Ж. " Электронное моделирование", № 4, 1996 г. 22-29.

[41] Симоненко В.П., Организация вычислительных процессов в параллельных вычислительных системах, М. "Информсвязь",1996, 150 стр.

[42] Трахтенгерц Э.Ф, Програмное обеспечение автомативизированных систем управления. М., Стастистика, 1974.

[43] Танаев В.С., Шкурба В.В., Введение в теорию расписаний, М. :Наука, 1975 г., 256 стр.

[44] Танаев В.С., Гордон В.С., Теория расписаний. Одностадийные системы, М. :Наука, 1984 г., 384 стр.

[45] Филипс Д., Гарсиа-Диас А., Методы анализа сетей. - М.:mМир, 1984.

[46] Форд Л. Р., Фалкерсон Д. Р., Потоки в сетях. -М.: Мир, 1966.

[47] Харари Ф., Теория графов. М. Мир, 1973.

[48] Adam T.L., Chandy K.M. and Dickson J.R., A comparision of List Scheduling for Parallel processing systems, Comm. ACM, Dec. 1974,pp685-690.

[49] Baker T.P., Stack Based Scheduling of Real-Time Processes, Journal on Real-Time Systems, Vol 3,N1,March 1991,pp67-99.

[50] Brendan Tangney, Donal O'Mahony, "Local Area Networks and Their Application", "Prentice Hall", New York, London, Toronto, Sydney, Tokyo,1988

[51] Berge C., Two Theorems in Graph Theory, Proc. National Acad. of Science USA, 43 (1957), 842-844.

[52] Blazevicz J., Drozdowski M., G. Schmidt, and D. De Werra, Scheduling independent multiprocessor tasks on a uniform k-processor system, Parallel Computer 20(1994),pp15-28.

[53] Blazevicz J., Drozdowski M., and J. Weglarz, Scheduling multiprocessor tasks to minimize schedule length, IEEE Trans. Computer C35(1986),pp389-393.

[54] Bultan T. and Aykanat C. , "A new mapping heuristic based on mean field annealing", *Journal of Parallel and Distributed Computing*, Vol. 16, n4, Dec 1992.

[55] Brendan Tangney, Donal O'Mahony, "Local Area Networks and Their Application", "Prentice Hall", New York, London, Toronto, Sydney, Tokyo,1988

[56] Butler R., Lusk E., Monitors, messages and clusters: The p4 parallel programming system, Technicfl Report Preprint MCS-P362-0493, Argone National Laboratory, Argone, 1993.

[57] Carriero N., Gelernter D., LINDA in context., Communication of the ACM 32(4): 444-458, 1989.

[58] Cheriyan J., Hagerup T. and Mehlhorn K., Can a Maximum Flow be Computed in O(nm) time?, to be presented at 17th ICALP,1990.

[59] Casavant Thomas L. and Kuhl John G., A taxonomy of Scheduling in General-Purpose Distributed Computing Systems, IEEE Transactions on Software Engeneering, Vol. 14,N2,1988, pp.141-154.

[60] Chang Y.C., Shin K.G., Optimal Load Sharing in Distributed Real-Time Systems, Journal of Parallel and Distributed Computing,Vol 19,1993,pp38-50.

[61] Coffman E. and Graham R., Optimal Scheduling for two-processor systems, Acta Information, Vol. 1,1972.

[62] Coli M. and Palazzari P. , Load Balancing with Internode Precedence Relations: a New Method for Static Allocation of DAGs into Parallel System. 1066-6192/96.

[63] Digital:, Digital's Networrs: an architecture with a future, Digital Equipment Corporation, 1984 (order no. EB 26013-42).

[64] Derigs U., Meier W., Implementing Goldberg's max-Flow-algorithm-A computational investigation. Z.oper. Res. A.-1989.-33,N 6.-p.383-403.

[65] Darte A., Two heuristics for Task Scheduling, Lab LIP-IMAG Ecole Norm. Super de Lyon,1991.

[66] Edmonds J., Karp R. M., Theoretical improvements in algorithmic efficiency for network flow problems, J. ACM.-Apr. 1972.-Vol. 19, N 2.-P. 248-264. (Combinatorial Structures and Their Applications. Gordon and Breach, New York, 1970, pp.93-96 (abstract presented at

Calgary International Conference on Combinatorial Structures and Their Applications, June 1969)).

[67] Eager D.L., Lazowska E.D., Zahorjan J., Adaptive Load Sharing in Homogeneous Distributed Systems, IEEE Trans. on Software Engineering, Vol. 12, (1986),pp662-675.

[68] Efe K., "Heuristic models for task assignment scheduling in distributed systems", *IEEE Computer*, June (1982)

[69] Elsadek A. A. and Wells B.E., "Heuristic model for task allocation in a heterogeneous distributed systems", *proc. of PDPTA'96,* Vol.2, (1996), pp659-671.

[70] El-Rewini H., Lewis T.G., "Scheduling Parallel tasks onto Arbitrary Target Machines", *Journal of Par. and Distr. Com.*,Vol.9,(1990),pp138-153.

[71] Evans D.J., Butt W.U.N., Dynamic Load Ballancing Using Task-Transfer Probabilities, Parallel Computing, Vol 19,1993,pp897-916.

[72] Feautrier Paul, Fine-grain Scheduling under Resource Constraints, Inter. Journal of Parallel Programming, Vol. 21, N 3, 1993, pp1-15.

[73] Feautrier Paul, Some efficient solutions to the affine scheduling problem,II, multidementional time, Int. Journal of Parallel Programming, Vol. 21, N 6, December 1992, pp389-420.

[74] Freund Richard F., Carter B.R., Watson Daniel, E. Keith, and F. Mirable, "Generational Scheduling for Heterogeneous Computing Systems", *proc. of PDPTA'96*, Vol.2, (1996), pp769-778.

[75] Flover J., Kolawa A., Bharadwaj S., The Express way to distributed processing, Supercomputing Review, pp. 54-55, 1991.

[76] Foster I., Designing and Building Parallel Programs, /book/ book/ ntm/1995.

[77] Gasperoni Franco and Schwiegelshohn Uwe, Scheduling loops on parallel processors: a simple algorithm with close to optimum performance, Parallel processing: CONPAR 92-VAPP V, Springer,1992, pp 625-636.

[78] Gelenbe E.,Kushwaha R., Dynamic Load Balancing in Distributed Systems, Mascost'94 IEEE Computer Society Press, pp.245-249, 1994.

[79] Gerasoulis A., Yang T., On the granularity and clustering of directed acyclic task graphs, IEEE Transactions on Parallel and Ditributed Systems, Vol 4, N. 6, June. 1993.

[80] Gerasoulis A and. Yang T., Scheduling Programs Task Graph on MIMD Architectures, Report DCS, Rutgers Uni.,1991.

[81] Gabow H. N., Scaling algorithms for network problems. J. omput. Syst. Sci. 31(1985), 148-168.

[82] Heywood T., Ranka S., A practical Hierarchical model of parallel computation, Journal of Parallel and Distributed Computing, 16, pp. 212-232, 1992.

[83] Herken R., The universal Turing Machine: a half-century survey, Oxford Press, 1988.

[84] Hwang K., Briggs F. A., Computer architecture and parallel processing, McGraw-Hill Booc C., 1989.

[85] Hou Edwin S.H., Ansari Nirwan and Ren Hong, A Genetic Algorithm for Multiprocessor Scheduling, IEEE Trans. on Par. and Distr. Systems, Vol.5, N2, Feb 1994, pp113-120.

[86] Homayoun N.,Ramanathan P., Dynamic Priority Scheduling of Periodic and Aperiodic Tasks in Hard Real-Time Systems, Journal on Real-Time Systems, Vol 6,N1, 1994,pp207-232.

[87] Hummel S.F., Schoneberg E., and Flynn E.L., Factoring: A Method for Scheduling Parallel Loops, Comm. Of the ACM 35(8), 1992,pp90-101.

[88] Hillis W.D., The Connection Machine, Mit Press Cambridg, Ma, 1985.

[89] Hoare C. A. R., Communication sequential processes, Prentice Hall Int., 1985

[90] Kim, C., Kameda, H., An Algorithm for Optimal Static Load balancing in Distributed Computer Systems, IEEE Trans on Comp., Vol.41(3),(1992),pp381-384.

[91] Kim S.J. and Brownne J.C., A general approach to mapping of parallel computation upon multiprocessor architectures, International Conference on Parallel Processing, Vol 3,1988.

[92] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., Optimization by simulated annealing, Science, Vol.220, N.4589, 13 May 1983.

[93] Kozielski S., Szczerbinski Z., Komputery rownolegle, Wydawnictctwa Naukowo-Techniczne Warszawa, 1993,201.

[94] Krajcoviech R. and Kotocova M., Executable File as Communication Link in Scheduling Process, PDPTA'96 Vol 2, 1996, pp713-725.

[95] Kremien O., Kramer J. : Methodical Analysis of Adaptive Load Sharing Algorithms, IEEE Trans.Par.Distr.Syst.3, pp747-760,(1992)

[96] Krommer A.R., Ueberhuber C.W., Architecture Adaptive Algorithms, Parallel Computing, Vol. 19 (1993),pp409-435.

[97] Kaufmann A., Introduction a la combinatorique en vue des applications. Dunod, Paris, 1968

[98] Liskov B., Schiefler R., Guardians and action: Linguistic support of robust distributed programs, Proceedings of the Ninth Symposium on Principles of Programming Languages, pp 7-19, 1982.

[99] Lam Monica, An effective scheduling technique for VLIW machines, Proc. Of the SIGPLAN'88 Conf.on Programming Language Design and Implementation, 1988, pp318-328.

[100] Lee C. , Hwang J. Chow Y., Anger F., Multiprocessor Scheduling with Interprocessor Communication delays, Operations Reaserch Letters, Vol 7,N3,pp141-147.

[101] Liu Jie, Saletore Vikram A., and Lewis Ted G.,Safe Self-Scheduling: A parallel loop scheduling Schem for Shared-Memory Multiprocessors, International Journal of Parallel Programming, Vol. 22, No. 6,1994, pp589-615.

[102] Liu J.W.S., Lin K.J., et al, Algorithms for Scheduling Imprecise Computations, IEEE Computer, Vol 24, N5, May 1991,pp58-68.

[103] Mechoso C.R., Farrara J. D., Achieving superlintar speedup on a Heterogeneous, Distributed system., IEEE Parallel&Distributed Technology, 2(2), 57-61, Summer, 1994.

[104] Minieka E., Optimization Algorithms for Networks and Graphs. Marcel Dekker, Inc., New York, 1978.

[105] Malloy Brian A., Lloyd Errol L., and Soffa Mary Lou, Scheduling DAG's for Asynchronous Multiprocessor Execution, IEEE on Parallel and Distributed Systems, Vol 5, ¹5(1994) , pp498-508.

[106] Mohr E. et al., Lazy Task Creation: A Technique for Increasing the Granularity of Parallel Programs, IEEE Tran. on Parallel and Distr. Systems,July 1991,pp264-280.

[107] Message Passing Interface Forum. Mpi: A message- passing interface standart. Computer Science Dept. Technicfl Report CS-94-230, University of Tennessee, Knoxville, TN, 1994.

[108] Neves Jose et al, A Software Agent Distributed System for Dynamic Load Balancing, Proc. of ESM'96,1996,pp80-84

[109] Ni L.M. and Wu C.E., Designing Tradeoffs for process Scheduling in Shared Memory Multiprocessor Systems,IEEE Tran. On Software Engineering 15(3),1989,pp327-334.

[110] P.H. Welch: "Parallel Hardware and Parallel Software : a Reconciliation", in the proceeding of the ZEUS'95 & NTUG'95, Conference, Linkoping< Sweden, 18-19 May, 1995; pp 287-301.

[111] Pham Hong Hang, Valery Simonenko, A new algorithm and simulation for task assignment in parallel distributed systems, Conference ESM96, Budapest, 1996, 95-99.

[112] Pham Hong Hang, Valery Simonenko, Adaptation of algorithm for job- resurce assignment in heterogeneous distributed systems, Conference PDPTA "96, Sunnyvale, Cflifornia USA, 1996,

[113] Phillips D., Garcia-Diaz A., Fundamentals of network analysis. Prentice-Hall, Inc., Englewood Cliffs, N. J., 1981.

[114] Pande Santosh, Agrawal Dharma P. and Jon Mauney, A scalable Scheduling Scheme for Funtional Parallelism on Distributed Memory Multiprocessor Systems, IEEE Tran. On Parallel and Distributed Systems, Vol.6, N 4, April 1995.

[115] Pham Hong Hanh, Valery Simonenko, Objective-Oriented Algorithm for Job Scheduling in Parallel Heterogeneous Systems, (11th International Parallel Processing Symposium (г. Женева Швейцария,1-5 Апреля 1997) 127-145 pp.

[116] Pham Hong Hanh and Simonenko Valery, Task Assignment for Scheduling Jobs and Resources in Parallel Distributed Systems, Journal Informatic and Kibernetic, Vol 12, N3, 1996,pp1-13.

[117] Phillips D., Garcia-Diaz A., Fundamentals of network analysis. Prentice-Hall, Inc., Englewood Cliffs, N. J., 1981.

[118] Polychronopoulos C. and Kuck D.J., Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel SuperComputers, IEEE Trans. On Computers 36(12),1987,pp1425-1439.

[119] Ramamritham K., Stankovic J.A., Zhao W., Distributed Scheduling of Tasks with Deadlines and Resource Requirments, IEEE TC-38,N8,August 1989,pp1110-1123.

[120] Rau B. R. and Glaeser C.D., Some scheduling techniques and an easily schedulable horizontal architecture for high-performence scientific computing, IEEE/ACM 14th Annual Microprogramming Workshop,1981.

[121] Riedl Reinhard and Richter Lutz, Classification of Load Distribution Algorithms, 1066-6192/96, IEEE Proceeding of PDP'96, pp404-413.

[122] Rost J. and Maehle E., A Distributed Algorithm for Dynamic Task Scheduling, Int. Journal of Parallel Programming, Vol 21,1996, pp628-639.

[123] Savage C., Maximum matching of trees. Inform. Process, Lett., Vol., 10 No. 4/5 (1980), p 202-205.

[124] Simonenko, Samofalov K.G., A new approach in solving problems in a distributed computer environment during dynamic schduling, First international ñonference on parallel processing and applied mathematics - Poland, 1994.

[125] Sakar V., Partition and Scheduling Parallel Programs for Execution on Multiprocessor, The MIT Press,1989.

[126] Seljak B.K., Task Scheduling Policies for Real-Time Systems, Microprocessors and Microsystems, Vol. 18,N9, 1994,pp501-511.

[127] Sha L. Rajkumar R, Lehotzky J.P. et al., Mode Change Protocols for Priority Driven Preemptive Scheduling, Journal on Real-Time Systems, Vol 1,1989,pp243-264.

[128] Shen X. and Reingold E.M., Scheduling on a hypercube, Informat. Processing Lett. 40(6)(1991),pp323-328.

[129] Tan M., Antonio J.K., et. al.,"Scheduling and data relocation for sequentially executed subtasks in a heterogeneous computing system", HCW'95,(1995),pp109-120.

[130] Tzen D.J. and Ni L.M., Trapezoid Self-Scheduling: A Practical Scheduling Scheme for Parallel Compilers, IEEE Trans. On Parallel and Distrib. Syst. 4(1),1993,pp87-98.

[131] Wang Y.T. , Morris R.J.T., Load Sharing in Distributed Systems, IEEE TC,March 1985,pp204-217.

[132] Wu M., Gajski D., A programming aid for hypercube architetures, The Journal of supercomputing, Vol 2,pp349-372,1988.

[133] Wu Shen Shen and Sweeting David, Heuristic Algorithm for Task Assignment and Scheduling in a Processor Network, Parallel Computing 20(1994),pp1-14.

[134] Yu-Kwong Kwok, Ishfad Ahmad, DynamicalCritical-Path Scheduling:An Effective Technique for Allocating Tasc Graphs to Multiprocessors, Parallel and distributed systems, V 7, ¹5, May 1996, 506-521 pp.

[135] Yu Ming-Shing., Yang Cheng-Hsing. A linear time algorithm for the maximum matching problem on cographs. BIT(Dan.).- 1993.- 33, N 3. p.420-433.

[136] Yang T. and Gerasoulis A., List Scheduling with and without Communication delays, Report DCS, Rutgers Uni.,1991.

[137] Yang T. and Gerasoulis A., Dominant sequence Clustering Heuristic Algorithm for Scheduling DAGs on Multiprocessor, Report of DCS, Rutgers Uni.,1991.

[138] Yue Kelvin K. and Lilja David J., Loop-Level Process Control: An Effective Processor Allocation Policy for Multiprogrammed Shared-Memory Multiprocessors, Journal of Parallel Programming, Vol. 21, 1996,pp182-199.

[139] Zhou, H.B., An effective approach for distributed program allocation, Journal of Parallel Algorithms and Applications, Vol. 2, No. 4, 1993.

[140] Zang X., Yan Y., Modeling and characterizing perallel computing performance on heterogeous networks of workstations., Proceedings of the Seventh IEEE Symposium on Parallel and Distributed Processing, 1995, 25-34.