УДК 004.021

Складання розкладів для мультипроцесорних систем в оперативному й автономному режимі роботи

Л. С. Глоба, Д. С. Лисенко
Національний технічний університет України "Київський політехнічний інститут", Україна

Ефективність роботи мультипроцесорної системи значною мірою залежить від методів побудови розкладу – способу за яким задачі розподіляються між процесорами з метою зниження часу роботи всієї системи. Ця стаття присвячена частині цього процесу – побудові розкладу на прикладі розробки програмного забезпечення для 4G базової станції.

Ключові слова: *мультипроцесорна система, складання розкладу, генетичний алгоритм.*

Эффективность работы мультипроцессорной системы в значительной мере зависит от метода построения расписания – способа распределения задач между процессорами с целью снижения времени работы всей системы. Данная статья посвящена части этого процесса – построению расписания на примере разработки программного обеспечения для 4G базовой станции.

Ключевые слова: *мультипроцессорная система, составление расписаний, генетический алгоритм.*

Efficiency of multiprocessor system usage is strongly dependent on methods of schedule design – the way of task distribution on each processor to decrease overall schedule time. This article is devoted to the part of this process - schedule design on example of software development for 4G base stations.

Key words: *multiprocessor system, schedule design, genetic algorithm.*

Вступ

Активний розвиток засобів телекомунікацій ставить перед розроблювачам задачу постійного оновлення існуючого програмного забезпечення (ПЗ) і розробки нового, що відповідає новим більш жорстким стандартам. Застосування мультипроцесорних систем особливо ускладнює це завдання.

У зв’язку із цим, метою представленої розробки є зниження часу розробки ПЗ телекомунікаційного обладнання, шляхом автоматизації визначення оптимальних рішень, пошуку помилок, спрощення налагодження й підвищення наочності опису функціонування ПЗ.

Для підвищення ефективності роботи ПЗ телекомунікаційного обладнання пропонується оригінальний метод складання ефективного розкладу для роботи ПЗ систем на кристалі (СНК). Апаратна платформа являє собою СНК, що має кілька типів різних процесорів (далі ресурсів). Алгоритм перетворення цифрового сигналу заданий у вигляді списку завдань із відносинами передування. Для кожного завдання вказується тип ресурсу, на якому він повинен бути виконаний й час необхідний для виконання.

При пошуку способу виконання алгоритмів ПЗ побудованого на основі пропонованого методу складання розкладів враховуються вимоги щодо максимально ефективного використання всіх доступних апаратних ресурсів. При цьому додатковою вимогою є мінімізація часу необхідного для пошуку такого рішення.

З метою зниження часу розробки ПЗ в складі запропонованого методу складання ефективного розкладу вирішуються наступні завдання:

- створення оптимального розкладу виходячи з вимог, які висуваються до ПЗ 4G базових станцій,
- мінімізація часу складання розкладу в оперативному режимі роботи.

2. Архітектура ПЗ СНК.

Архітектура розглянутої СНК представлена на рис.1. Розглянемо докладніше процес її роботи. Генерація списків завдань відбувається на макрорівні, потім завдання надходять на рівень прикладних програм нижньої підсистеми або фізичний рівень. Після обробки на фізичному рівні, списки завдань обробляються ПЗ, що працює на системному рівні. ПЗ, використовуючи інтерфейси операційної системи (ОС), забезпечує виконання отриманих списків завдань із використанням усіх доступних ресурсів, що знаходяться на рівні апаратних абстракцій.

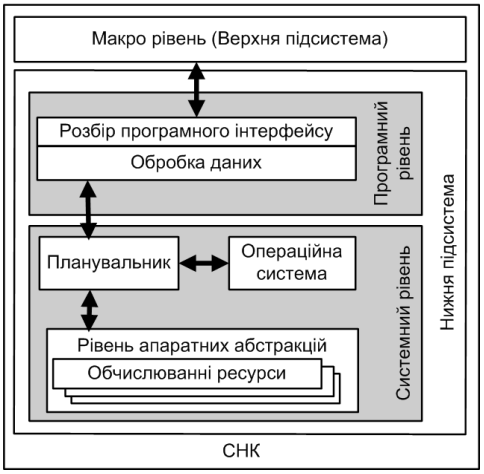

Рис.1. Структурна схема ПЗ СНК.

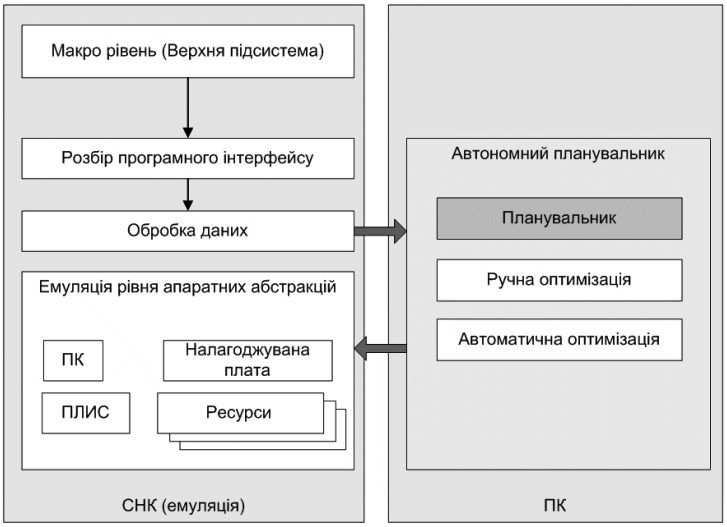

Рис.2. Емуляція роботи ПЗ у режимі налагодження.

Робота ПЗ у режимі реального часу ускладнює налагодження й не дозволяє застосовувати ресурсномісткі алгоритми складання розкладів. Тому аналіз списку завдань виконується автономно на програмному комплексі чи персональному комп'ютері (ПК). При цьому обробка завдань може бути виконана як на реальних ресурсах так і з використанням різних рівнів емуляції (рис.2)

Розглянемо роботу ПЗ в оперативному режимі (рис.3). Планувальник при наявності вільних ресурсів вибирає з отриманого списку завдань відповідні й направляє їх на виконання. Коли черга завдань не містить завдань, робота планувальника завершується.

Для налагодження роботи ПЗ і оцінки якості складеного розкладу в процес обробки завдань додається етап протоколювання. Аналіз отриманого протоколу і його оптимізація виконується на ПК в автономному режимі (рис. 4) у два етапи. На першому етапі після завершення всіх завдань, що отримуються від ПЗ, яке виконується на СНК знімається протокол роботи. Після аналізу даних протоколу, у разі отримання висновків про необхідність внесення змін у список завдань, обладнання перезапускається й на другому етапі, у момент, коли список завдань готовий до обробки, в нього вносяться необхідні зміни. Після обробки зміненого списку завдань знову формується протокол роботи, він аналізується з точки зору корисності внесених змін.

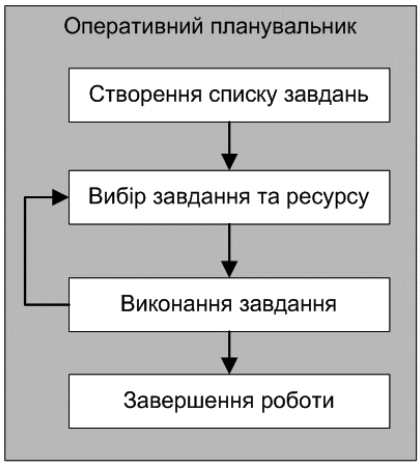

Рис.3. Схема роботи планувальника в оперативному режимі.

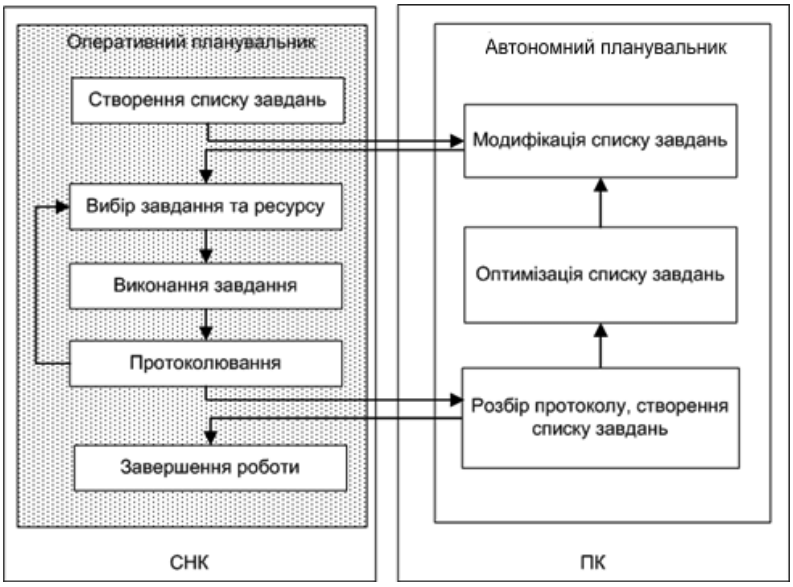

Рис.4. Спільна робота автономного й оперативного планувальників.

Припущення при розв'язанні задачі.

При аналізі й розв'язанні задачі використовувалися наступні припущення, що випливають зі специфіки предметної області.

1. Виконання завдання не переривається
2. Виконання завдання не розділяється між ресурсами
3. Кожен ресурс може виконувати тільки одне завдання (немає поділу часу)
4. Критерієм оптимальності складеного розкладу є його довжина. Якщо завдання W_i виконується за час, C_i , то загальний час виконання $\Sigma_{i=0}^{N} C_i$ для N завдань.

5. Кожне завдання має відомий заздалегідь час виконання.
6. Кожному завданню співвіднесений список завдань, які повинні бути виконані раніше його.
7. Система має відоме заздалегідь, скінченне число ресурсів кожного типу.
8. На початку роботи, планувальник має відомості про всі завдання, що становлять розклад. Під час виконання завдання не видаляються й не створюються нові.
9. Час, необхідний на завантаження або вивантаження завдання й підготовку ресурсу до роботи, не враховується.
10. В оперативному режимі, задача складання розкладу виконується на окремому ресурсі й час розрахунків розкладу не повинний впливати на загальний час обробки сигналу.
11. Кожне завдання має рівень пріоритету, який призначений попередньо. При інших рівних завдання, що має більш високий пріоритет, повинно бути виконано раніше.

3. Побудова математичної моделі обчислювального процесу.

Математична модель обчислювального процесу може бути представлена в вигляді графів «завдання - вершини» та «завдання - дуги».

Вхідні дані.

Вхідні дані для алгоритму перетворення цифрового сигналу в процесі роботи БС складаються з наступного набору даних:

- Множина завдань W і задане в ньому відношення передування: кожному із завдань $\mu \in W$ зіставляється деяка підмножина P її попередників. Причому для деяких завдань $P = \varnothing$.
- Кожному завданню зіставляється ціле ненегативне значення пріоритету $\ddot{}_{\mu}$ і індекс типу ресурсу, якому вона відповідає r_{μ} . Час виконання завдання t_{μ}. Унікальний у межах списку індекс k_{μ} використовується для запобігання невизначеності при сортуванні завдань.
- Множина доступних ресурсів R задана заздалегідь і містить у собі дані про кількість доступних ресурсів кожного типу $R = \{R_0, R_1, ...R_i\}$.

Вхідні дані можна представити у вигляді графу $G = (S_n, S_a)$, де $S_n = \{n_i, i = 1..N\}$ – множина вузлів, які представляють завдання, та $S_a = \{(n_i, n_j) | n_i \to n_j\}$ – скінченна множина дуг, які представляють відношення передування завдань. При цьому кожному завданню n_i ставиться у відповідність час виконання w_i.

Формально описати складання розкладу можна як множину Φ функцій g, які дозволяють відобразити граф G на архітектуру ПЦОС $S_p = \{p_k, k = 1..P\}$,

де P – кількість ресурсів. Функція $g(n_i)$ повертає множину ресурсів p_k, на яких n_i виконується. Далі, $\forall\, g \in \Phi\, \exists\, \overline{g}$, де $\overline{g}(p_k)$ множина вузлів, призначених на процесор p_k функцією g. Аналогічно $\forall\, g \in \Phi\, \exists\, S_g$, де S_g - множина допустимих розкладів S, які можуть бути отримані за допомогою g. Таким чином, $S_g \subseteq S_t$, де S_t - множина існуючих допустимих відображень G на S_p - так званий *простір рішень*. Для заданого розкладу $s \in S_g$, якщо $n_i \notin \overline{g}(p_k)$, то $s(n_i, p_k)$ не визначене, в іншому випадку воно відображає час початку виконання n_i на ресурсі p_k.

Використовуючи наведені визначення, розклад s, який створюється g є допустимим, якщо наступні вирази істинні для усіх пар вузлів $(n_i, n_j) \in S_a$.

$$\forall p_l \in g(n_i) : \text{якщо } p_l \in g(n_i) \text{ то } s(n_i, p_l) + w_i \le s(n_j, p_l),$$

$$\text{якщо } p_l \notin g(n_i) \quad \text{то } \exists\, p_k \in g(n_i) \,\big|\, s(n_i, p_k) + w_i + s(n_j, p_l)$$

$$\forall\, p_k \in S_p, \forall\, n_i \in S_n, n_j \in \overline{g}(p_k) :$$

$$\text{якщо } n_i \ne n_j \wedge s(n_j, p_k) \ge s(n_i, p_k) \text{ ,то } s(n_j, p_k) \ge s(n_i, p_k) + w_i$$

Граф «завдання - вершини».

Найбільш очевидним є представлення списку завдань у вигляді орієнтованого графа $\langle M, N, T \rangle$ де M - завдання, N – умови передування, T - відображення множини у прямий добуток $M \times M$, що ставить у відповідність кожній залежності $j \in N$ упорядковану пару завдань, перша з яких повинна передувати другий.

Крім візуальної оцінки топології списку завдань, граф $G_1 = \langle M, N, T \rangle$ дозволяє провести перевірку графа на відсутність контурів. Завдання, що становлять контур, не можуть початися й чекають один одного.

Граф «завдання - дуги».

На практиці представлення списку завдань, у вигляді графа «завдання - вершини» не завжди зручно. Як приклад можна привести наступний граф, отриманий на основі реальних даних, що й становить собою частина розрахунків фрейму WIMAX (рис.5).

На ньому практично неможливо простежити залежності між завданнями, і він не придатний до використання.

Щоб уникнути таких незручностей, було вирішено представляти граф залежностей між завданнями, вважаючи завдання дугами. У цьому випадку вершинами графа стають події початку й закінчення завдань.

$$G_2 = \langle M', N', T' \rangle$$

Застосування таких перетворень до наведеного вище графа завдань дає результат більш наочний і зручний у використанні (рис. 6).

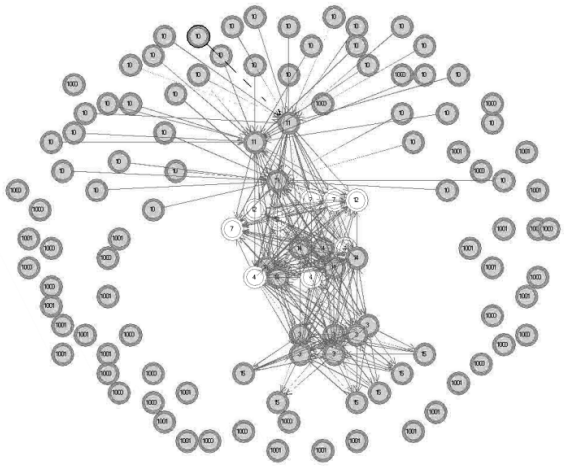

Рис.5. Приклад графа "завдання - вершини" з більшою кількістю зв'язків.

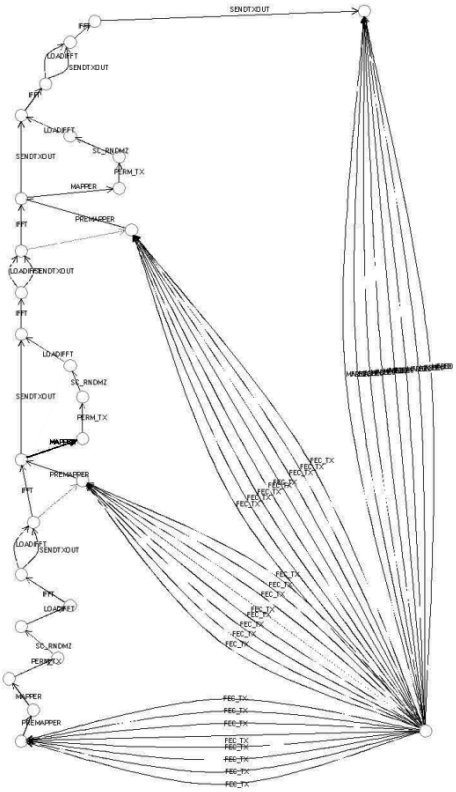

Рис. 6 Приклад графа "завдання - дуги".

Для оцінки стану виконання списку завдань у будь-який момент часу використовується діаграма Гантта (стрічкова діаграма).

4. Аналіз обчислювального процесу.

Для аналізу обчислювального процесу використовуються 4 основних методи: пошук контурів, побудова графа «завдання - дуги», приведення графа до ярусно паралельного вигляду й розрахунок критичного шляху (КШ).

Пошук контурів.

Пошук контурів здійснюється шляхом перебору, у якім будується постійно нарощуваний шлях . Він або замикається на вершину, що вже міститься в ньому або зупиняється в тупиковій вершині, у цьому випадку, дуга до такої вершини може бути вилучена з розгляду й зі шляху. Відзначимо, що формалізувати процес розмикання контурів неможливо. Апріорно припустиме будь-яке упорядкування завдань, і єдина можливість вирішення колізії - експертний аналіз проблеми поза рамками математичної моделі.

Побудова графа «завдання - дуги».

Будемо вважати, що на цьому етапі граф завдань вільний від контурів. У цьому випадку першим етапом буде побудова транзитивного замикання $\overline{P}$ відносини P для знаходження всіх попередників для кожного із завдань.

Для визначення завдань, що мають спільний набір попередників, розглянемо сукупність множин $\overline{P}_\mu, \mu \in W$ і вилучимо із цієї сукупності повтори, у результаті виходить деякий набір множин $®_~, ® \in A$, і кожному $\mu \in W$ буде відповідати індекс $~$, такий що $\overline{P}_\mu = ®_~(\mu)$.

Для розбивки множини завдань на групи розглянемо для кожного $®_~$ розбивку $\Pi_~ = ®_~ \cup (\frac{W}{®_~})$. Розбивка $\overline{\Pi}_~ = \{ ° \}_{\in B}$, дрібніше кожного з розбивок співмножників, отже кожне з множин $®_~$ є об'єднанням деякого набору множин $°$. Позначимо через $B(~) \subset B$, набір індексів для яких $®_~ = \cup_{\in B(~)} °$. Тому що об'єднання всіх множин $°$ збігається з W і ці множини не перетинаються, то кожний елемент W належить рівно одному з них. Іншими словами, кожному $\mu \in W$ однозначно зіставляється індекс $(\mu) \in B$,що $\mu \in °_{(\mu)}$.

Уважаючи на те, що обрані нами множини A і B диз'юнктивні, будуємо граф $\langle M, N, T \rangle$, де $M = A \cup B$; $N = W \cup F$; $T(\mu) = (~(\mu), \ (\mu))$. Щоб зібрати разом усі фрагменти отриманого двочасткового графа, необхідні додаткові дуги. Визначимо множину фіктивних дуг F як дуги відповідні до всіх пар $(~, \ )$, при чому початок дуги відповідає $\in B(~)$,а кінець $~ \in A$.

Отриманий граф може бути істотно спрощений, якщо додержуватися наступним правил:

- Якщо існує обхідний шлях для дуги $j_f \in j$ не утримуючий дуги j_f , те така дуга може бути вилучена із графа.

- Якщо якась дуга $j_f \in F$ єдина дуга вихідна з вершини j_f або вхідна у вершину, те така дуга може бути вилучена шляхом склеювання її початку й кінця в одну вершину.

Приведення графа до ярусно паралельного вигляду.

При оцінці топології графа, важливо подати інформацію в зручному для користувача вигляді для цього граф приводиться до ярусно паралельного вигляду. Це така розбивка вершин графа на яруси при якім:

1. Вершини першого ярусу не мають попередніх вершин, а вершини останнього ярусу наступних
2. Вершини будь-якого ярусу не мають попередніх у групі вершин наступного ярусу.
3. У кожному ярусі повинна бути хоча б одна дуга з попереднього ярусу.
4. Вершини в межах одного ярусу не з'єднуються дугами.

Така розбивка завжди можлива, і проводиться з використанням наступного алгоритму:

1. Будується матриця суміжності вершин.
2. У черговий ярус вибираються вершини з нульовим напівступенем заходу, вони відповідають нульовим стовпцям. Рядки відповідні до обраних стовпців зануляються.
3. Другий крок повторюється доти поки не будуть упорядковані всі вершини.

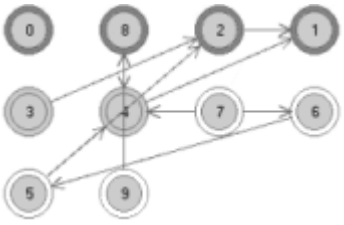

Рис. 7:Приклад графа "завдання - вершини" до приведення до ЯПВ.

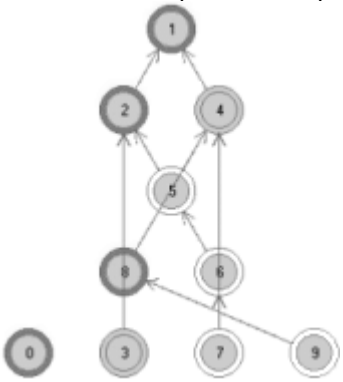

Рис. 8:Приклад графа "завдання - вершини" після приведення до ЯПВ.

Розрахунок критичного шляху.

Граф отриманий у результаті перетворення списку завдань може бути використаний для одержання важливих для розрахунків кількісних характеристик, насамперед пов'язаних із часом виконання.

Позначимо час виконання кожного завдання $\mu \in W$ як $t_\mu > 0$ При розгляді графа «завдання - дуги», для фіктивних завдань уважаємо $t_\mu = 0$ Сума часу завдань для шляху назвемо довжиною шляху. Пошук критичного шляху, це необхідна частина аналізу списку завдань при перевірці можливості паралельного виконання й оцінки часових характеристик розглянутого списку.

Алгоритм пошуку критичного шляху (КШ) схожий з алгоритмом Дейкстры з поправкою на те, що шукається шлях не мінімальної а максимальної довжини.

Алгоритм пошуку критичного шляху.

Кожній події $i \in M$ зіставляється число $з(i)$,називане раннім настанням події i і рівне найбільшій довжині шляхи від початкової події до i.

Ранні початки подій можуть бути визначені зі співвідношення

$$з = \max\{t_j + \circ з(\textit{початок } j) \,|\, \textit{кінець } ji\},$$

при чому для початкової події $i_0, з(i_0) = 0$.

Усі розрахунки не можуть бути завершені раніше ніж відбудеться заключна подія, у такий спосіб довжина критичного шляху це мінімальний час завершення розрахунків.

Аналогічно алгоритму Дейкстры розіб'ємо множину вершин $M = M_0 \cup M_1 \cup M_2$, де M_0 - множина вершин з розрахованим часом настання ранньої події, M_1 - множина вершин з розрахованим часом настання ранньої події й M_2 - множина вершин для яких обчислення ще не починалися. Крім того кожній вершині $i \in M$ зіставляється ціле ненегативне число $r(i)$ - кількість ще не переглянутих дуг з кінцем в i та число $з(i)$ – раннє настання події i.

Початковий стан:

$$M_0 = \varnothing,\ M_1 = \{i\},\ M_2 = \frac{M}{M_1},\ M = 0,\ r = \left|\{\, j \,|\, \textit{кінець } j = i\}\right|,\ j \in M$$

Крок алгоритму:

1. Якщо $M_1 = \varnothing$, то обчислення завершуються.
2. Інакше, вибирається будь-яка дуга $i_1 \in M_2$, яка переноситься в M_0 .
3. Для кожної дуги j вихідної з i_1 , та її кінця i_2 , порівняти $з(i_2)$ і $з' = t(j) + з(i_1)$ та збільшити $з(i_2)$ до $з'$ якщо $з'$ більше. Зменшити $r(i_2)$ на 1 і при $з(i_2) = 0$ перенести i_2 з M_2 в M_1 .

Аналогічно раннім настанням подій можуть обчислюватися пізні настання. Взявши в якості вихідних моментів для кожної події критичний час проекту й уважаючи цей час уже обчисленим для заключної події, обчислюємо заново пізні настання для всіх інших подій по формулі: $V = \min\{-t_j + V(\textit{кінець } j) \,|\, \textit{початок } ij\}$.

Знаходження пізніх моментів аналогічно знаходженню ранніх початків.

Для кожного завдання обчислимо величину $d = V(кінець\ i) - t(j) - v(\ початок\ j)$ і назвемо її резервом часу для завдання Очевидно, що завдання лежачі на критичному шляху мають нульовий резерв часу.

Нижче наведений приклад розрахунків критичного шляху.

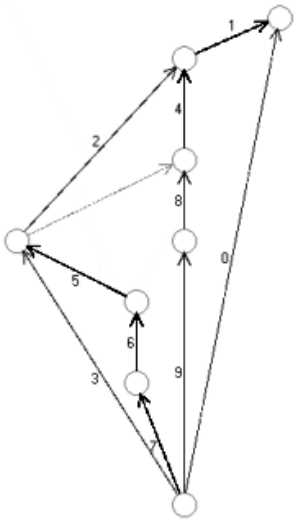

Рис. 9: Приклад розрахунків критичного шляху.

Довжина критичного шляху 46 тактів.

Завдання, що лежать на критичному шляху, відзначені сірим.

Табл. 1: Приклад розрахунку критичного шляху.

Ім'я	Вага	Раннє настання події	Пізнє настання події	Резерв часу
0	10	0	36	36
8	10	12	26	14
2	2	2	10	8
1	**2**	**0**	**0**	**0**
3	10	4	36	32
4	**10**	**2**	**2**	**0**
7	**2**	**44**	**44**	**0**
6	**2**	**42**	**42**	**0**
5	**30**	**12**	**12**	**0**
9	10	22	36	14

З наведеного графа видно, що завдання 1,4,5,6 і 7 лежать на критичному шляху й відсутність затримок у них виконанні важливо для складання оптимального розкладу. Так само можна відзначити, що завдання 2 має невеликий резерв часу й слід звернути увагу на забезпечення його своєчасного виконання.

5. Алгоритми побудови розкладу

Жорсткі вимоги, пропоновані до алгоритмів побудови розклади в оперативному режимі роботи, унеможливлюють застосування евристик. Замість цього необхідно використовувати алгоритми, що вимагають мінімальної кількості процесорного часу й ресурсів. Тому запропоновано 4 типу алгоритмів. Алгоритми 1, 2 і 3 можуть використовуватися в оперативному режимі, хоча алгоритм 3 вимагає більшу кількість ресурсів. Через високі вимоги до обчислювальних ресурсів алгоритм 4 може бути реалізований тільки на ПК. Питання про вибір використовуваного алгоритму в кожному конкретному випадку може бути вирішено залежно від кількості доступних апаратних ресурсів.

Алгоритм 1: «Швидкий».

Алгоритм, наведений нижче використовує мінімальну кількість ресурсів для складання розкладу, він запускається щораз при старті планувальника й при завершенні завдання.

1. Для кожного завдання $\mu \in W$, де W - розглянута множина завдань, що відповідає наступним умовам :
   a. Призначена на конкретний тип ресурсу H_k .
   b. Що має список залежностей P_μ .
2. Множина завдань W поділяється на K підмножин відповідних до кожного типу ресурсів.
3. Для $k = \overline{1, K}$ множин, M_k завдань.
4. Поки $M_k^i, i = \overline{1, N_k}$ обробляємо елементи множини k .
5. Якщо ресурс P_k^i вільний.
6. Нехай $j = 1$.
7. Поки $j < M_k$.
8. Якщо завдання T_j не має активних залежностей .
9. Виконати завдання T_j на ресурсі P_k^i .
10. Інакше, наступне i .
11. Інакше, наступне j .
12. Наступний k .

Алгоритм 2: «Критичний шлях» - модифікація «Швидкого» алгоритму використанням результатів розрахунків критичного шляху.

Будь-які затримки на критичному шляху приводять до збільшення загального часу розкладу. Щоб не допустити цього встановимо пріоритет кожного завдання рівним резерву часу. Це приведе до того, що завдання, що лежать на критичному шляху, будуть виконуватися в першу чергу, що знижує ймовірність затримок для критичних завдань.

Розрахунки критичного шляху для вихідних даних виконується заздалегідь за допомогою автономного ПЗ.

1. Для кожного завдання $\mu \in W$, де W - розглянута множина завдань, що відповідає наступним умовам :
   a. Призначена на конкретний тип ресурсу H_k .
   b. Що має список залежностей P_μ .
   c. Що має приорітет Π_μ
2. Множина завдань W поділяється на K підмножин відповідних до кожного типу ресурсів.
3. Для $k = \overline{1, K}$ множин, M_k завдань.
4. Поки $M_k^i, i = \overline{1, N_k}$ обробляємо елементи множини k .
5. Якщо ресурс P_k^i вільний.
6. Нехай $j = 1$.
7. Поки $j < M_k$.
8. Якщо завдання T_j не має активних залежностей .
9. Виконати завдання T_j на ресурсі P_k^i .
10. Інакше, наступне i .
11. Інакше, наступне j .
12. Наступний k .

Алгоритм 3: « Критичний шлях з резервуванням» - модифікація алгоритму « Критичний шлях» веденням резервування ресурсів для критичних завдань.

Розглянемо окремий випадок вихідних даних (Таблиця 2). Усі завдання використовують той самий тип ресурсів.

Табл. 2: Вихідні дані.

Номер завдання	Час виконання	Залежності
1	10	-
2	5	1
3	5	1
4	9	-
5	5	2,3
6	3	4

Рис. 10: Граф "завдання - дуги".

Граф відповідний цим вихідним даним наведений на малюнку (рис. 10). Критичний шлях позначений червоним. Застосування алгоритму 2 для випадку із двома доступними ресурсами дає результат в 22 такту й розклад наступного вигляду (рис. 11).

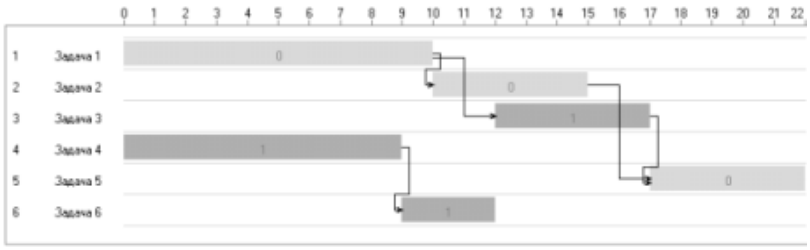

Рис. 11: Розклад побудоване за допомогою алгоритму 2.

Нескладно помітити, що виконання завдання 6 відразу після завдання 4 призвело до затримки виконання завдання 3, що лежить на критичному шляху і як наслідок до збільшення часу розкладу в цілому. Щоб уникнути таких ситуацій і знизити ймовірність затримок на критичному шляху введемо в алгоритм етап резервування ресурсів під критичні завдання. Таким чином, алгоритм здобуває наступний вид:

1. Для кожного завдання $\mu \in W$, де W - розглянута множина завдань, що відповідає наступним умовам :
   a. Призначена на конкретний тип ресурсу H_k .
   b. Що має список залежностей P_μ .
   c. Що має приорітет Π_μ
2. Множина завдань W поділяється на K підмножин відповідних до кожного типу ресурсів.
3. Для $k = \overline{1,K}$ множин, M_k завдань.
4. Поки $M_k^i, i = \overline{1,N_k}$ обробляємо елементи множини k .
5. Якщо ресурс P_k^i вільний.
6. Нехай $j = 1$.
7. Поки $j < M_k$.
8. Якщо завдання T_j не має активних залежностей .

9. Виконати завдання T_j на ресурсі P_k^i .
10. Інакше, наступне i .
11. Інакше, наступне j .
12. Наступний k .

Результат роботи алгоритму представлений нижче (рис. 12).

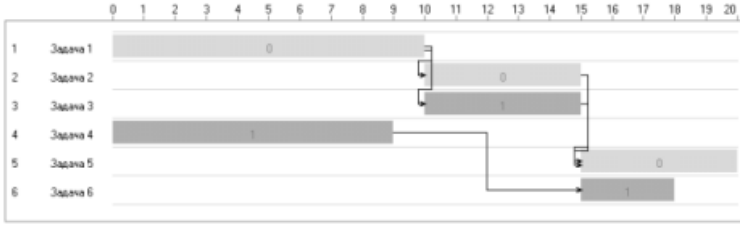

Рис. 12:Розклад, побудований за допомогою алгоритму 3.

У результаті застосування резервування, на момент запуску завдання 3, вільний ресурс був знайдений, і загальний час розкладу скоротився до 20 тактів.

Алгоритм 4: Генетичний.

Оскільки задача складання розкладу є Np-повною, точний алгоритм вирішення даної проблеми за кінцевий час не може бути знайдений. Для знаходження субоптимального вирішення був використаний генетичний алгоритм (ГА) – один з видів стохастичних алгоритмів, який містить у собі евристичні методи, що моделюють еволюційні механізми відбору й пошуку можливих рішень. Цей метод добре зарекомендував себе в вирішення задач складання розкладів [8],[9],[10].

Опис алгоритму.

Схема алгоритму наведена на рисунку 13.

Рис. 13: Схема генетичного алгоритму.

Створення початкової популяції.

Створення початкової популяції проводиться шляхом випадкового розміщення пріоритетів завданням і внесенням випадкової невеликої кількості додаткових залежностей. Так само дозволяє знизити кількість ітерацій до

одержання субоптимального результату внесення в популяцію індивідуума створеної на основі алгоритму КШ. Однак для наочності результатів подібний індивідуум при проведенні розрахунків не додавався.

Цільова функція.

У якості цільової функції ГА використовувався час, що вийшов у результаті складання розкладу за допомогою «швидкого» алгоритму розглянутого раніше. У такий спосіб цільова для списку з завдань може бути представлена у вигляді $\Theta(W, P_0, P_1, ...P_n)$

У цьому випадку кожне з розглянутих розкладів презентовано у вигляді множини пріоритетів завдань, що входять у розклад і набору додаткових залежностей P'_μ

$\Theta(¨_0, ¨_1, ...¨_n, k_0, k_1, ...k_n, r_0, r_1, ...r_n, P_0, P_1, ...P_n, P'_0, P'_1, ...P'_k)$, де k - число додаткових залежностей

Індекс завдання k_μ уведено для того щоб уникнути невизначеності при виборі черговості запуску завдань із однаковим пріоритетом. Виходячи з опису цільової функції розглянемо два типи генів – пріоритет завдання ¨ і додаткова залежність P множина цих параметрів становить хромосому.

Мутація.

Під мутацією розуміється введення одиничної зміни в хромосому індивідуума. У розглянутому випадку це відповідає додаванню або видаленню додаткового зв'язку або зміні пріоритету завдання на 1. Вибір типу мутації здійснюється випадковим образом із імовірністю 0.5 . Щоб уникнути появи петель у графі при додаванні нових залежностей необхідно враховувати ярус вузла. Для цього граф завдань приводиться до ярусно-паралельної формі й при додаванні нової залежності використовується множина завдань, що мають більш високий номер ярусу стосовно поточної.

Вибір батьків.

На етапі генерації наступного покоління необхідно забезпечити ефективний відбір батьківських індивідуумів для схрещування. Виходячи з рекомендацій до створення ГА була обрана наступна функція розподілу батьківських індивідуумів:

80% відібраних з 20 % популяції, що мають найкращу цільову функцію, 10% з 10% популяції, що мають найгірше значення цільової функції й 10% інших.

Схрещування.

При схрещуванні для одержання індивідуума нащадка використовується комбінація генів батьківських індивідуумів, кількість генів від кожного з батьківських індивідуумів визначається випадковим образом.

Збереження елітних індивідуумів.

Щоб уникнути регресії при роботі алгоритму, частина індивідуумів, що мають найкращий результат, фіксується й не піддається мутаціям. Дослідним шляхом установлене, що їх число повинне бути невеликим приблизно 0.05% від розміру популяції, але не менше 1.

Завершення алгоритму.

Час роботи алгоритму може, обмежуються за часом, кількості розглянутих

поколінь, або ознаці виродження популяції – коли значення цільової функції середнє по популяції починає дорівнювати кращому значенню.

Вибір оптимальних параметрів генетичного алгоритму.

В процесі роботи було помічено, що час потрібний ГА для знаходження субоптимального результату значною мірою знаходиться в залежності від наступних параметрів : розмір популяції, кількість пар для схрещування та кількість мутацій. Для знаходження оптимальних значень цих параметрів було виконане дослідження, результати якого представлені на малюнку нижче (рис. 14).

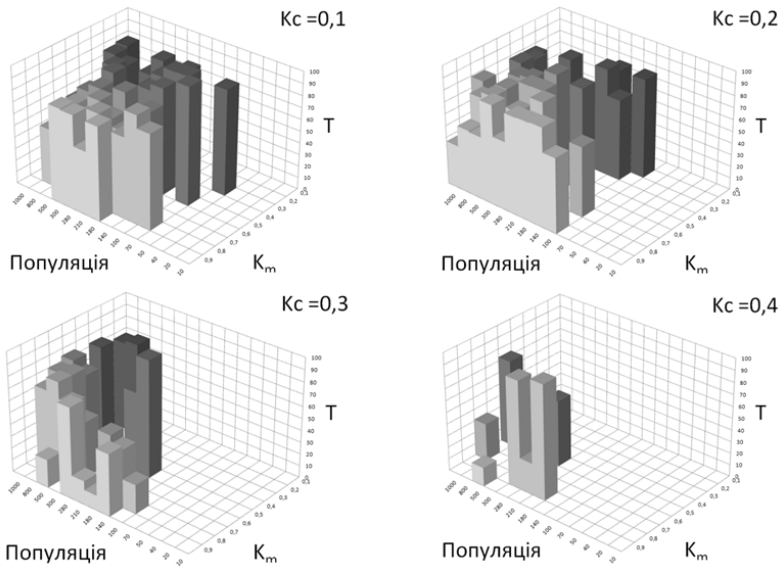

Рис. 14: Результати вибору оптимальних параметрів генетичного алгоритму.

На малюнку позначено K_c – коефіциент кількості пар для схрещування відносно розміру популяції, K_m - коефіциент кількості мутацій відносно розміру популяції. Для оцінки ресурсомісткості (тобто часу роботи) ГА використовулася кількість розрахунків цільової функції у процентах до її мінімального значення (T). На діаграмах позначені лише результати в яких субоптимальне значення було досягнуте за менше чим 800 000 розрахунків цільової функції. Таких результатів для $K_c > 0.4$ не було знайдено жодного.

Для проведення дослідження був вибраний зразок №1. Таким чином, для роглядаємої множини зразків були вибрані наступні параметри ГА: розмір популяції 70, кількість пар для схрещування 7 кількість мутацій 35 за одну ітерацію алгоритму.

Оптимізація складання розкладу в оперативному режимі роботи планувальника.

Для оптимізації довгі розклади використовується ГА у зв'язку з тим що його результатом роботи є не тільки складений розклад але й параметри індивідуума

з найкращою цільовою функцією: набір пріоритетів для кожного завдання й список додаткових зв'язків. Застосування цих параметрів до вихідного списку завдань викличе зменшення загального часу розкладу при використанні алгоритму 2 в оперативному режимі.

Розглянемо цей процес на прикладі наведеному в (Таблиця 1). Як було показано вище в даному окремому випадку з використанням алгоритму 2 створюється неоптимальний розклад довжиною 22 такту. Результат роботи ГА представлений у таблиці (Таблиця 2).

Табл. 2:Результати роботи ГА.

Номер завдання	Пріоритет	Додаткові залежності
1	9	
2	9	
3	9	
4	8	
5	9	
6	0	1

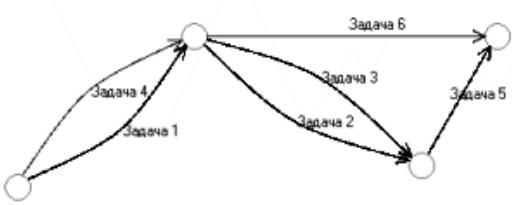

Рис. 15: Результати роботи ГА

Граф «завдання - дуги», відповідний до знайденого рішення, наведений вище на рисунку 15. Використовуючи алгоритм 2 до зміненого списку завдань, ми одержуємо рішення аналогічне отриманому за допомогою алгоритму 3 (див. рис. 12) менше на 2 такта в порівнянні з вихідним.

6. Результати експериментів.

Запропонований алгоритм реалізований мовою програмування C# у рамках створення комплексу «Offline Sceduler». Були проведені експерименти з метою перевірки ефективності алгоритму й настроювання його основних параметрів: імовірності появи мутації, імовірності застосування схрещування, розміру популяції й числа елітних індивідуумів .

У якості прикладів використовувалася серія з 32 списків завдань виникаюча в ході формування WIMAX фрейму. Останній список завдань був узятий із процедури формування фрейму LTE і характеризується високою можливістю до распаралеливаню. Усі списки використовують два типи різних ресурсів. Усі зразки вихідних даних містять від 97 до 1087 завдань і від 116 до 87317 зв'язків.

Диагама (рис. 16) показує результати розрахунків формування розкладу при кількості ресурсів першого типу 2 і другого типу 4 для наступних алгоритмів

«швидкий», «критичний шлях» (КШ), «критичний шлях з резервуванням» (КШР) і генетичний. Для генетичного алгоритму використовувалася популяція розміром 200 і розрахунки закінчувався при досягненні 150ї ітерації. Результати наведені відносно «швидкого» алгоритму.

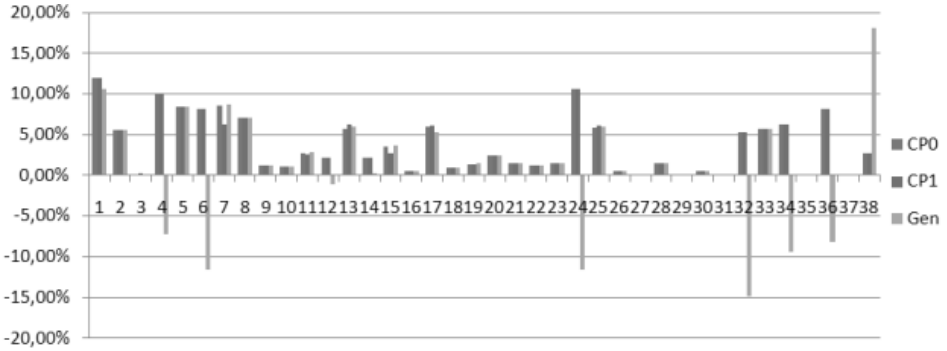

Рис. 16: Результати розрахунків трьох, що перевіряються алгоритмів стосовно "швидкого".
Ресурсів 1го типу 2, ресурсів 2го типу 4.

Як видно для зразка 4 генетичний алгоритм показує найгірший результат, але при збільшенні популяції до 1000 і кількості ітерацій до 500 значення порівнюються (рис. 16). Краще рішення був знайдене на 378 ітерації й надалі залишалося постійним при проведені ще 2000 ітерацій.

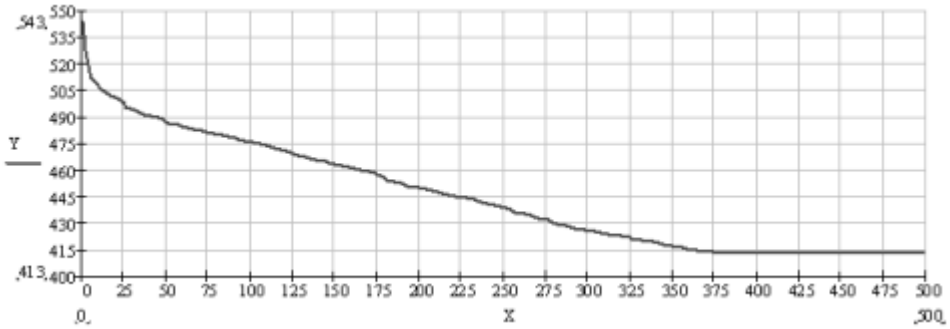

Рис. 17: Зразок 4, популяція 1000, найкращий результат 413.

Розрахунки, що дали схожі результати були проведені й для інших зразків.

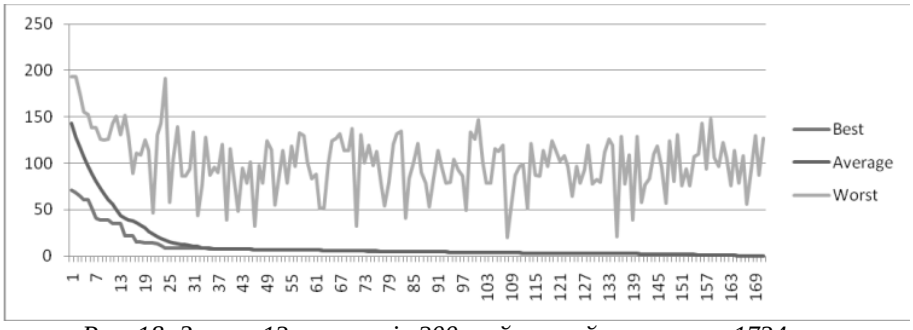

Рис. 18: Зразок 13, популяція 200, найкращий результат 1734.

Для зразка 13 так само наведений гірший і середній результат по популяції на всіх етапах обчислень (рис.17).

Таким чином, можна зробити висновок про те, що серед чотирьох розглянутих алгоритмів генетичний дозволяє одержати найкращий результат при непорівнянному часі розрахунків, наприклад для зразка 4 рішення 413 було отримано за допомогою алгоритму « критичний шлях» менше ніж за секунду, знаходження цього ж результат за допомогою генетичного алгоритму зайняло більш 3х годин.

Обчислення проведені для кількості ресурсів першого типу 4 і другого типу 4 дозволяють зробити висновок про те, що при збільшенні кількості доступних ресурсів результати застосування різних алгоритмів розрахунків розклади прагнуть до значення критичного шляху.

Необхідно відмітити що особливо гарні результати навіть при невеликому числі ітерацій генетичний алгоритм показує на списках завдань із великою можливістю до распараллелюваню.

Висновки

1. Для вирішення завдання складання розкладу запропоновані й програмно реалізовані
    - Математичні моделі обчислювального процесу у вигляді графів «завдання - вершини» і «завдання - дуги»
    - Запропоновані методи аналізу й перетворення цих моделей, що дозволяє візуально оцінити стан обчислювального процесу.
    - Чотири типи алгоритмів, два придатні до реалізації в режимі реального часу й два більш ресурсномісткі для побудови субоптимального розкладу.
2. Реалізований генетичний алгоритм, що дозволяє не тільки скласти розклад але й зробити зміну розглянутого списку завдань, що дозволяє зменшити загальний час розкладу для алгоритмів працюючих в оперативному режимі.
3. Проведені експериментальні дослідження для більш ніж 30 варіантів черги завдань і різною кількістю доступних ресурсів. Виявлені параметри ГА які дають найкращий результат за мінімальний час для розглянутої множини завдань.

ЛИТЕРАТУРА

1. Э.Г. Коффман. Теория расписаний и вычислительные машины. Москва "Наука", 1984 г.
2. Mohammadi and S. Aki, “Scheduling Algorithms for Real-Time Systems“, TR No.2005-499 Queen’s University Publication. 2005.
3. Srinivasan, P. Holman, J. Anderson, and S. Baruah, “The Case for Fair Multiprocessor Scheduling”, 2002.
4. Krishna and K. Shin, “Real Time Systems”, McGraw-Hill, 1997.

5. И.В. Романовский. Дискретный анализ. Учебное пособие для студентов, специализирующихся по прикладной математике и информатике. СПб. "Невский диалект", 1999 г.
6. С.И. Зуховский, И.А. Радчик Математические методы сетевого планирования. Москва "Наука", 1965 г.
7. Phu D. Hoang and Jan M. Rabaey “Scheduling of DSP Programs onto Multiprocessors for Maximum Throughput” IEEE TRANSACTIONS ON SIGNAL PROCESSING, VOL. 41, NO. 6, JUNE 1993 p.2225
8. Robert W. Amphlett and David R. Bull “GENETIC ALGORITHM BASED DSP MULTIPROCESSOR SCHEDULING”, Centre for Communications Research,University of Bristol, Bristol, UK.
9. Peter Koch, Niels Larsen, Thomas Bauer, Ole Ejlersen “GENETICAS: A Multi-DSP Scheduling Technique Based on Genetic Algorithms” .The DSP Research Group Institute for Electronic Systems, Aalborg University
10. Pavel Osmera, Bronislav Lacko, Masopust Petr “Parallel Evolutionary Algorithms”, Pruceedinpr 2003 IEEE International Sympusium on Computational Intelligent in Robtics and Automatiun July 16-20,2003, Kobe, Japan
11. Shrutisagar C Sridhara, Krishnan N “Double Abstraction Level Heuristic Power Optimization for Digital Signal Processors Using GA’s”, Mepco Schienk Engg.

Надійшла у першій редакції 21.09.2011, в останній – 19.10.2011.