

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ
ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри

_____ І.А. Жуков
(підпис)

« ____ » _____ 2020 р.

ДИПЛОМНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТР
ЗА СПЕЦІАЛЬНІСТЮ 123 «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»

Тема: _____
«Комп'ютер з мінімальним енергоспоживанням»

Виконавець: _____
студентка групи КС-221 Драч Валерія Ігорівна

Керівник: _____
доцент, к.т.н., Єфимець Валентин Микитович

Нормоконтролер: _____
(підпис) _____
Малярчук В.О.
(ПІБ)

Засвідчую, що у дипломній роботі немає
запозичень праць інших авторів без
відповідних посилань

Студент _____
(підпис) _____
Драч В.І.
(ПІБ)

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютерних систем та мереж

Спеціальність 123 “Комп'ютерна інженерія”

ЗАТВЕРЖУЮ

Завідувач кафедри

І.А. Жуков

(підпис)

« ____ » _____ 2020 р.

ЗАВДАННЯ на виконання дипломної роботи

Драч Валерії Ігорівни

(прізвище, ім'я, по батькові)

1. Тема роботи: «Комп'ютер з мінімальним енергоспоживанням»

затверджена наказом ректора від « 25 » вересня 2020 року № 1793/ст

2. Термін виконання роботи: з 05.10.2020 р. _____ по 30.12.2020 р. _____

3. Вихідні дані до роботи: Використання методів зменшення енергоспоживання для створення енергосвідомих комп'ютерних систем.

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):

1) Розробка різних енергетичних моделей та інтеграція з існуючими симуляторами та вимірювальними інструментами;

2) Огляд методів DPM націлених на зменшення споживання енергії під час роботи шляхом вибіркового вимкнення або уповільнення компонентів;

3) Огляд методів, які можуть бути використані для зменшення споживання енергії, необхідної для (бездротового) зв'язку поза комп'ютером.

5. Перелік обов'язкового графічного матеріалу:

Презентація PowerPoint

6. Календарний план-графік

№ з/п	Завдання	Термін виконання	Підпис керівника
1	Ознайомитись з постановкою задачі дипломного проектування	25.09.2020-30.09.2020	
2	Вивчити спеціальну літературу і технічну документацію	01.10.2020-10.10.2020	
3	Провести аналіз наявних енергетичних моделей	11.10.2020-15.10.2020	
4	Написати розділ 1 дипломного проекту	16.10.2020-25.10.2020	
5	Провести аналіз методів управління живленням	26.10.2020-05.11.2020	
6	Написати розділ 2 дипломного проекту	06.11.2020-25.11.2020	
7	Провести дослідження можливості зниження енергії у спілкуванні	26.11.2020-30.11.2020	
8	Написати розділ 3 дипломного проекту	31.11.2020-10.12.2020	
9	Підготувати графічний демонстраційний матеріал	11.12.2020-20.12.2020	
10	Захистити дипломний проект	21.12.2020-22.12.2020	

7. Дата видачі завдання: «25» вересня 2020 р.

Керівник дипломної роботи: _____ Єфимець В.М.
(підпис)

Завдання прийняв до виконання: _____ Драч В.І.
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Комп'ютер з мінімальним енергоспоживанням», 93 с., 15 рис., 9 табл., 86 літературних джерел.

Об'єкт дослідження: енергоспоживання системи на рівні центрального процесора, системного рівня та паралельних системних рівнів.

Мета роботи: розробка та дослідження методів зменшення енергоспоживання для створення енергосвідомих комп'ютерних систем.

Методи дослідження: аналіз можливостей зниження енергоспоживання охоплюючи різні системні компоненти, розробка різних енергетичних моделей та інтеграція з існуючими симуляторами та вимірювальними інструментами. Установлено, що завдяки зменшенню енергоспоживання портативні пристрої за один той самий час роботи здатні оброблювати значно більший обсяг завдань, що істотно впливає на попит користувачів на такі системи.

Проведене дослідження є значущим, беручи до уваги поточну ситуацію в світі так як воно допоможе знизити енергоспоживання, що позитивно вплине як на екологічну ситуацію у світі, так і на можливості користувачів при використанні портативних пристроїв.

Результати магістерської роботи рекомендуються використовувати під час проведення наукових досліджень та в практичній діяльності кафедри комп'ютерних систем та мереж для постановки курсу «Архітектура комп'ютерів».

ЕНЕРГЕСПОЖИВАННЯ, ПОТУЖНІСТЬ, ЖИВЛЕННЯ, РІВНІ СИСТЕМИ,
МЕРЕЖА, ЕНЕРГЕТИЧНА МОДЕЛЬ, АРХІТЕКТУРА КОМП'ЮТЕРА, ПРОТОКОЛ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ.....	7
ВСТУП.....	9
РОЗДІЛ 1 АНАЛІЗ ТА ОПТИМІЗАЦІЯ ПОТУЖНОСТІ З ВИКОРИСТАННЯМ ЕНЕРГЕТИЧНИХ МОДЕЛЕЙ.....	14
1.1. Моделі енергоспоживання на рівні процесора.....	16
1.1.1. Модель енергії центрального процесора на рівні циклу.....	16
1.1.2. Модель енергії процесора на рівні інструкції.....	18
1.2. Повні енергетичні моделі на системному рівні.....	20
1.2.1. Апаратна енергетична модель стану.....	21
1.2.2. Енергетична модель на основі процесів.....	22
1.2.3. Специфічна для компонентів енергетична модель.....	24
1.3. Моделі енергії на рівні взаємозв'язку в паралельних системах.....	25
Висновки за розділом.....	26
РОЗДІЛ 2 МЕТОДИ ДИНАМІЧНОГО УПРАВЛІННЯ ЖИВЛЕННЯМ (<i>DPM</i>).....	28
2.1. <i>DPM</i> на рівні процесора.....	30
2.1.1. Зменшення активності комутації.....	31
2.1.2. <i>Clock gating</i>	32
2.1.3. Динамічне масштабування напруги (<i>DVS</i>).....	33
2.1.3.1. Планувальник на основі інтервалів.....	37
2.1.3.2. Методи міжзадачних завдань для додатків у режимі реального часу.....	38
2.1.3.3. Прийоми внутрішньозадачних задач для додатків у режимі реального часу.....	42
2.2. Повна система системного рівня <i>DPM</i>	44
2.2.1. Політика <i>DPM</i> на основі апаратних пристроїв.....	45
2.2.2. Програмні політики <i>DPM</i>	48

2.2.2.1. Розширений інтерфейс налаштування та живлення (ACPI)....	49
2.2.2.2. DPM на основі додатків.....	51
2.2.2.3. DPM на основі операційної системи.....	52
2.3. Паралельний DPM на системному рівні.....	53
2.3.1. Апаратні технології DPM.....	54
2.3.1.1. Узгоджене динамічне масштабування напруги (CVS).....	54
2.3.1.2. DPM на основі мережових взаємозв'язків.....	55
2.3.2. Програмні технології DPM.....	56
2.3.2.1. DPM на основі бар'єрних операцій.....	56
2.3.2.2. DPM з балансуванням навантаження.....	57
Висновки за розділом.....	57
РОЗДІЛ 3 ЗНИЖЕННЯ ЕНЕРГІЇ У СПІЛКУВАННІ.....	59
3.1. Джерела споживання енергії.....	59
3.2. Стек мережевого протоколу.....	61
3.3. Протоколи MAC.....	66
3.3.1. S-MAC.....	67
3.3.2. T-MAC.....	69
3.3.3. D-MAC.....	71
3.3.4. B-MAC.....	72
3.3.5. B-MAC +.....	74
3.3.6. X-MAC.....	75
3.3.7. WiseMAC.....	76
3.4. Розкладання системи.....	78
3.5. Мережі короткого радіусу дії низької потужності.....	79
Висновки за розділом.....	80
ВИСНОВКИ.....	82
СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ.....	86

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

<i>ACPI</i>	–	<i>Advanced Configuration and Power Interface</i>
<i>APM</i>	–	<i>Advanced Power Management</i>
<i>AVS</i>	–	<i>Automatic Voltage Scaler</i>
<i>BIOS</i>	–	<i>Basic Input/Output Services</i>
<i>BMAC</i>	–	<i>Berkley Media Access Control</i>
<i>CCA</i>	–	<i>Clear channel assesment</i>
<i>CMOS</i>	–	<i>Complementary Metal Oxide Semiconductor</i>
<i>COPPER</i>	–	<i>Compiler-controlled continuous Power-Performance</i>
<i>CVS</i>	–	<i>Coordinated Voltage Scaling</i>
<i>DLS</i>	–	<i>Dynamic Link Shutdown</i>
<i>DPM</i>	–	<i>Dynamic Power Management</i>
<i>DVS</i>	–	<i>Dynamic Voltage Scaling</i>
<i>DVS-DM</i>	–	<i>DVS with Delay and Drop rate Minimizing</i>
<i>DVS-PD</i>	–	<i>DVS with Predicting Decoding time</i>
<i>Flops</i>	–	<i>Floating-point Operation Per Second</i>
<i>FTRS</i>	–	<i>Future request to send</i>
<i>GOP</i>	–	<i>Group Of Pictures</i>
<i>CTS</i>	–	<i>Clear to Send</i>
<i>I/O</i>	–	<i>Input/Output</i>
<i>IVS</i>	–	<i>Independent Voltage Scaling</i>
<i>JVM</i>	–	<i>Java Virtual Machine</i>
<i>LCD</i>	–	<i>Liquid-Crystal Display</i>
<i>MPEG</i>	–	<i>Moving Pictures Expert Group</i>
<i>NAV</i>	–	<i>Network allocation vector</i>
<i>PLL</i>	–	<i>Phase-Locked Loop</i>
<i>POSE</i>	–	<i>Palm Operating System Emulator</i>

<i>RTL</i>	–	<i>Register Transfer Language</i>
<i>RTS</i>	–	<i>Request to Send</i>
<i>RWEC</i>	–	<i>Remaining Worst-case Execution Cycles</i>
<i>SPEC</i>	–	<i>Standard Performance Evaluation Corporation</i>
<i>VOVO</i>	–	<i>Varry-On/Varry-Of</i>
ПК	–	Персональний комп'ютер
ЦП	–	Центральний процесор
ОС	–	Операційна система
АП	–	Апаратне забезпечення
ПЗ	–	Програмне забезпечення
ТА	–	Тайм-аут

ВСТУП

Споживання енергії є обов'язковою умовою існування людства. Наявність доступної для споживання енергії завжди було необхідною умовою для задоволення потреб людини, збільшення тривалості та поліпшення умов його життя.

У сучасному світі енергетика є основою розвитку базових галузей промисловості, що визначають прогрес суспільного виробництва. В усіх промислово розвинених країнах темпи розвитку енергетики випереджали темпи розвитку інших галузей.

У той же час енергетика - одне з джерел несприятливого впливу на навколишнє середовище і людину. Вона впливає на атмосферу (споживання кисню, викиди газів, вологи і твердих частинок), гідросферу (споживання води, створення штучних водоймищ, скиди забруднених і нагрітих вод, рідких відходів) і на літосферу (споживання викопних палив, зміна ландшафту, викиди токсичних речовин) .

Незважаючи на зазначені фактори негативного впливу енергетики на навколишнє середовище, зростання споживання енергії не викликало особливої тривоги у широкій громадськості. Так тривало до середини 70-х років, коли в руках фахівців виявилися численні дані, що свідчать про сильний антропогенний тиск на кліматичну систему, що таїть загрозу глобальної катастрофи при неконтрольованому зростанні енергоспоживання. З тих пір жодна інша наукова проблема не привертає такої пильної уваги, як проблема справжніх, а особливо майбутніх змін клімату.

Вважається, що однією з головних причин цієї зміни є енергетика. Під енергетикою при цьому розуміється будь-яка область людської діяльності, пов'язана з виробництвом і споживанням енергії.

Ні для кого не секрет, що сфера інформаційних технологій з кожним роком розвивається все більше. Тільки за 2019 рік за світовими даними було придбано 71,78 млн. комп'ютерів, що на 4,8% більше ніж за даними 2018 року. Тобто можна побачити, що з кожним роком кількість купівель тільки зростає. Можна тільки уявити скільки енергії витрачається на забезпечення роботи даних пристроїв, вже не кажучи про

кількість енергії необхідної для їх виробництва. Вчені всього світу зараз в пошуках методів і варіантів для зменшення енергоспоживання комп'ютерних систем, це не тільки позитивно вплине на екологічну ситуацію в світі, але і збільшить попит користувачів на товар, тому виробники також в цьому зацікавлені.

Метою та завданням даної дипломної роботи є розробка та дослідження методів зменшення енергоспоживання для створення енергосвідомих комп'ютерних систем. Аналіз можливостей зниження енергоспоживання охоплюючи різні системні компоненти, розробка різних енергетичних моделей та інтеграція з існуючими симуляторами та вимірювальними інструментами дозволить зрозуміти на що саме потрібно звертати увагу при розробці комп'ютерів з мінімальним енергоспоживанням та як кожен з цих методів вплине на кінцевий результат.

Об'єктом дослідження роботи є енергоспоживання системи на рівні центрального процесора, системного рівня та паралельних системних рівнів.

Темі роботи неможливо було б розвивати не торкнувшись терміну «архітектура комп'ютера». Зазвичай під архітектурою ПК розуміють їх організацію з точки зору користувача, тобто весь комплекс програмних і апаратних засобів, що забезпечує процес програмування і рішення програм на комп'ютерах. Опис включає в себе інформацію про користувацьких можливостях програмування, системи команд, структури центрального процесора і організації пам'яті, а також опис апаратних засобів, що входять до складу ПК, набору зовнішніх пристроїв і способів зв'язку з ними.

Інновації та вдосконалення вже давно зроблені в архітектурах комп'ютерів та систем, щоб суттєво збільшити обчислювальну потужність. Удосконалення напівпровідникової технології дає можливість включити мільйони транзисторів на дуже малій матриці та встановити їх на дуже високій швидкості. Архітектура та технологія системного програмного забезпечення також пропонують значні покращення продуктивності, використовуючи паралелізм у різних формах. Хоча попиту на ще більш потужні комп'ютери буде заважати фізика обчислювальних систем, таких як обмеження напруги та швидкість перемикання [33], більш критичною та безпосередньою

перешкодою є споживання енергії та відповідні теплові проблеми та проблеми надійності [23] . Це стосується не тільки портативних систем низького класу, а й висококласних системних конструкціям.

Оскільки такі портативні системи, як портативні комп'ютери та стільникові телефони, заряджають акумулятори, зменшення енергоспоживання для продовження часу їх роботи є однією з найбільш важливих специфікацій продукту. Це також є проблемою для дизайнерів високого класу, оскільки велике споживання енергії підвищує температуру, що погіршує продуктивність та надійність. У деяких екстремальних випадках для цього потрібна дорога окрема енергетична установка, як у *Earth Simulator* [18], яка досягає пікової продуктивності 40 *Tflops*, але розсіює 5 МВт потужності.

У цій роботі подано комплексний огляд методів аналізу та оптимізації потужності. Методи енергоефективних комп'ютерних систем можна розділити на два типи: офлайн-аналіз енергії та техніки динамічного управління енергією. Методи офлайн-аналізу потужності засновані на аналітичних енергетичних моделях, які вбудовані в існуючі симулятори, орієнтовані на продуктивність, щоб отримати інформацію про потужність та продуктивність та допомогти архітекторам систем вибрати найкращі параметри системи під час проектування. Динамічна потужність управління (*DPM*) схеми контролюють робоче навантаження системи та адаптують поведінку системи для економії енергії. Ці методи - це динамічні схеми виконання, що працюють на різних рівнях комп'ютерної системи. Вони включають схеми динамічного масштабування напруги (*DVS*), які регулюють напругу живлення та робочу частоту процесора для економії енергії в режимі очікування [27, 68]. Подібна ідея може бути застосована до пристроїв вводу-виводу, відстежуючи їх діяльність та вимикаючи або уповільнюючи їх, коли попит на ці пристрої низький [7, 14, 28]. Інша можливість збереження енергії під час виконання - це коли система має більше одного ресурсу одного типу, який зазвичай знаходиться в паралельних та мережевих кластерних системах. У цьому випадку

застосування схеми *DPM* узгоджено, а не її застосування до окремих ресурсів, може краще керувати цілою системою.

Розділ 1 обговорює кілька енергетичних моделей та відповідні методи аналізу та оптимізації потужності, інтегровані в існуючі середовища моделювання. Ці енергетичні моделі охоплюють різні рівні системи з різним ступенем деталізації та точності, що включає методи аналізу потужності на рівні центрального процесора, системного рівня та паралельних системних рівнів.

У розділі 2 представлені різні технічні та програмні технології *DPM*, які також відрізняються деталізацією та точністю. Дрібнодисперсний моніторинг та управління енергією можливий у меншому масштабі, але це може бути неможливим у більшому масштабі через відповідні накладні витрати на збір інформації та прийняття рішень, пов'язаних з владою. Тому в цьому розділі представлені різні технології *DPM* на рівні центрального процесора, системного рівня та паралельних системних рівнів.

Розділ 3 розглядає методи, які можуть бути використані для зменшення споживання енергії, необхідної для (бездротового) зв'язку поза комп'ютером. Мінімізація споживання енергії є завданням, яке вимагатиме мінімізації внесків зв'язку та обчислень, внесення відповідних компромісів між ними.

Зменшення енергоспоживання стало головним викликом у проектуванні та експлуатації сучасних комп'ютерних систем. Актуальність цієї теми підтверджена світовими фахівцями, праці яких і стали поштовхом для її розгляду. Раціональність людства до використання ресурсів зростає, бо вже майже кожен розуміє які наслідки можуть бути, якщо нічого не зміниться.

Мій особистий внесок при написанні даної дипломної роботи – аналіз можливостей зниження енергоспоживання охоплюючи різні системні компоненти, розробка різних енергетичних моделей та інтеграція з існуючими симуляторами та вимірювальними інструментами.

Зібрані та структуровані матеріали роботи зможуть стати навчальним посібником для студентів, які вирішать зануритися у тему впливу компонентів системи на енергоспоживання.

РОЗДІЛ 1

АНАЛІЗ ТА ОПТИМІЗАЦІЯ ПОТУЖНОСТІ З ВИКОРИСТАННЯМ ЕНЕРГЕТИЧНИХ МОДЕЛЕЙ

Розсіювання потужності стало основним обмеженням у проектуванні процесорів та комп'ютерних систем. Оптимізація енергії, як і продуктивність, вимагає ретельного проектування на декількох рівнях архітектури системи. Першим кроком на шляху оптимізації споживання енергії є розуміння джерел споживання енергії на різних рівнях. Були розроблені різні енергетичні моделі та інтегровані з існуючими тренажерами або вимірювальними інструментами для забезпечення точної оцінки потужності, яка може бути використана для оптимізації конструкції системи.

Розділ 1.1 описує енергетичні моделі на основі процесора, які оцінюють споживання енергії на рівні циклу або інструкції. У розділі 1.2 розглядаються системні енергетичні моделі, що вивчають споживання енергії як апаратних, так і програмних компонентів. Нарешті, розділ 1.3 націлений на багатопроцесорні та кластерні енергетичні моделі. Ці дослідження, зокрема, зосереджуються на взаємозв'язку системи, оскільки енергетичні показники окремих процесорів або вузлів можуть бути оцінені на основі методів, описаних у розділах 1.1 та 1.2. У табл. 1.1 узагальнено ці підходи, що базуються на енергетичній моделі.

Таблиця 1.1

Таксономія методів аналізу потужності з використанням енергетичних моделей

Тип	Рівень деталізації	Енергетичні моделі	Інструменти моделювання	Розділ
1	2	3	4	5

Продовження таблиці 1.1

1	2	3	4	5
ЦП	Рівень циклу або RTL	Модель на основі щільності потужності або ємності моделювання на рівні циклу	<i>PowerTimer</i> [6], <i>Wattch</i> [7] та <i>SimplePower</i> [56]	1.1.1
	Рівень інструкції	Енергетична модель на основі інструкцій із вимірюванням кількості інструкцій	Профілі живлення для <i>Intel 486DX2</i> , <i>Fujitsu SPARClite '934</i> [51] та <i>PowerPC</i> [39]	1.1.2
Система	Рівень апаратних компонентів	Модель станів для функціонального моделювання	<i>POSE</i> [13]	1.2.1
	Рівень програмних компонентів	Модель, заснована на процесі, з дискретизацією, керованою часом та енергією	Дискретизація керована часом, <i>PowerScope</i> [16], та дискретизація керована енергією [12]	1.2.2
	Рівень апаратних та програмних компонентів	Компонентні енергетичні моделі для повного моделювання системи	<i>SoftWatt</i> побудований на симуляторі системи <i>SimOS</i> [23]	1.2.3
Паралельна система	Рівень архітектури взаємозв'язку	Модель бітової енергії для моделювання рівня бітів	Інструмент на основі <i>Simulink</i> [57]	1.3
		Енергетична модель, що базується на повідомленнях, для моделювання мережі взаємозв'язку	<i>Orion</i> , симулятор енергоефективних мереж взаємозв'язку [53]	1.3

1.1. Моделі енергоспоживання на рівні процесора

Споживана центральним процесором енергія є основною частиною загального енергоспоживання комп'ютерної системи і, отже, є основною метою аналізу енергоспоживання [9, 10, 49, 65, 70]. Кілька моделей живлення були розроблені та інтегровані в існуючі симулятори продуктивності, щоб дослідити споживання енергії центрального процесора як на основі функціональних блоків, так і на процесорі в цілому. Ці аналізи базуються на двох рівнях абстракції: рівень циклу (або рівень передачі реєстру) та рівень інструкцій, як описано у наступних двох підрозділах, відповідно.

1.1.1. Модель енергії центрального процесора на рівні циклу

Енергоспоживання процесора можна оцінити за допомогою симуляторів архітектури на рівні циклу. Це робиться шляхом виявлення активних (або зайнятих) блоків або блоків рівня мікроархітектури протягом кожного циклу виконання модельованого процесора [9, 10, 70]. Потім ці статистичні дані про використання ресурсів за циклами можуть бути використані для оцінки споживання енергії. Енергетична модель, що описує, як кожен компонент або блок споживає енергію, є ключовим компонентом будь-яких симуляторів рівня циклу. Рис. 1.1 ілюструє блок-схему високого рівня силових симуляторів рівня циклу.

Брукс та ін. представив два типи енергетичних моделей для симулятора *PowerTimer* [6]:

- модель енергії на основі щільності потужності використовується для компонентів, коли доступні детальні вимірювання потужності та площі;
- аналітичні енергетичні моделі використовуються для решти компонентів процесора.

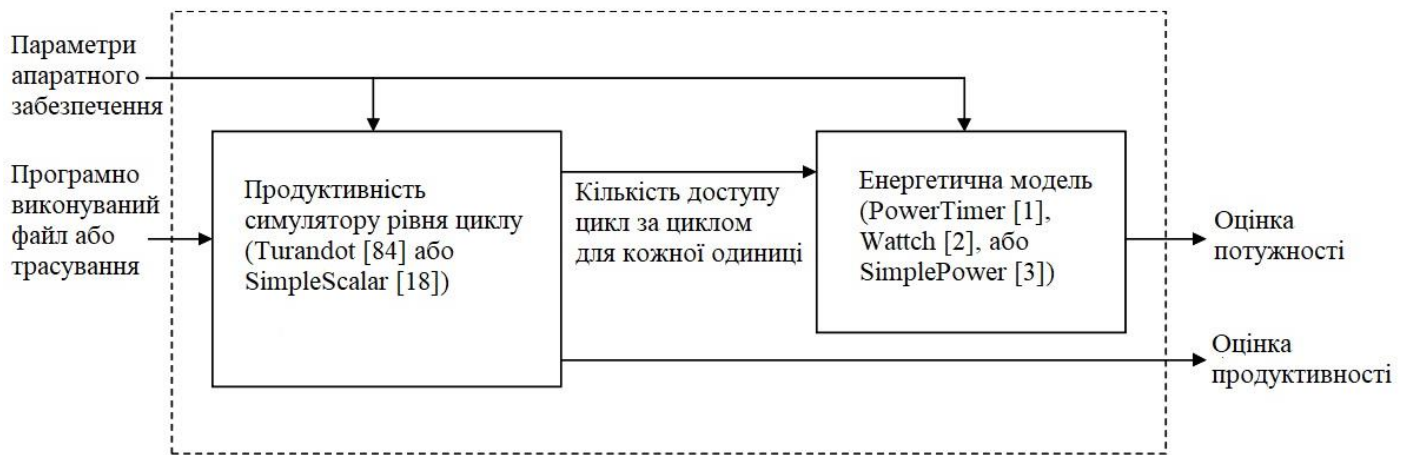


Рис. 1.1. Структурна схема силового симулятора рівня циклу

Аналітичні рівняння формують енергетичні характеристики з точки зору параметрів проектування на рівні мікроархітектури, таких як розмір кеша, довжина конвеєру виконання завдань, кількість регістрів тощо. Ці два типи енергетичних моделей використовувались у поєднанні із загальним, параметризованим, неупорядкованим суперскалярним симулятором процесора під назвою *Turandot* [36]. Використовуючи *PowerTimer*, можна вивчити компроміси між енергоефективністю та різними конфігураціями системи з різними розмірами кеш-пам'яті, черги обробки завдань, реєстрів перейменувань та таблиць прогнозування гілок, що допоможе у побудові мікроархітектур, що усвідомлюють енергію.

Wattch [7] та *SimplePower* [56] - це два інших інструменти контролю енергоспоживання на рівні процесора, засновані на *SimpleScalar* [8], який є найпопулярнішим симулятором мікроархітектури. У *Wattch* енергетичні моделі залежать від внутрішніх ємностей ланцюгів, що складають кожну одиницю процесора. Кожен змодельований блок потрапляє в одну з наступних чотирьох категорій: структури масивів, пам'яті, комбінаційної логіки, дротів та тактової мережі. Для кожної категорії використовується інша модель потужності, яка інтегрована в симулятор *SimpleScalar* для забезпечення різноманітних показників, таких як потужність, продуктивність, енергія та обмежувач напруги. У табл. 1.2 наведено витрати енергії на різні компоненти при вимірах, а також від симулятора *Wattch*.

Порівняння розподілів потужності між вимірюванням (*Alpha 21264*) та аналітичною енергетичною моделлю в симуляторі *Wattch*

Апаратна структура	Вимірювання (<i>Alpha 21264</i>)	Аналітична модель (<i>Wattch</i>)
Кеш	16,1%	15,3%
Неправильна логіка випуску	19,3%	20,6%
Пам'ять	8,6%	11,7%
Блок управління пам'яттю	10,8%	11,0%
Блок виконання з плаваючою точкою	10,8%	11,0%
Мережа тактової частоти	34,4%	30,4%

SimplePower, з іншого боку, базується на енергетичній моделі, чутливої до переходу, де кожен змодельований функціональний блок має власну ємність комутатора для кожного можливого вхідного переходу [56]. Потім це використовується для обчислення потужності, споживаної в певному функціональному блоці на основі вхідного переходу під час виконання заданої інструкції. *SimplePower* використовується для оцінки впливу архітектурної модифікації, а також впливу техніки оптимізації компілятора високого рівня на потужність системи. Приклади використання *SimplePower* включають селективну технологію керованого конвеєра для зменшення ємності комутатора шляхів передачі даних, контуру та перетворення даних для зменшення потужності системи пам'яті та реєстрацію повторного маркування для економії потужності на шинах даних [56].

1.1.2. Модель енергії процесора на рівні інструкції

На відміну від методів точного розрахунку, методи аналізу потужності з грубими рівнями підрахунку на рівні інструкцій оцінюють загальну вартість енергії програми,

додаючи енергію, спожиту під час виконання інструкцій програми [65, 9]. Витрати енергії за інструкцією, які називаються базовими, можуть бути виміряні для окремих інструкцій для цільового процесора. Однак існує додаткове споживання електроенергії завдяки "взаємодії" між послідовним виконанням команд внутрішнього струму, спричиненими головним чином ефектами конвеєру виконання завдань та кешу. Базові витрати на окремі інструкції та енергетичні витрати на ефекти між інструкціями визначаються на основі експериментальної процедури з використанням програми, що містить кілька екземплярів цільової інструкції (для вимірювання базової вартості) та чергується послідовність інструкцій (для ефектів між інструкціями витрати). Табл. 1.3 ілюструє підмножину базових витрат на *Intel 486DX2* та *Fujitsu SPARClite'934* [51]. Подібне дослідження також було проведено для мікропроцесора *PowerPC* [39].

Таблиця 1.3

Базові витрати на процесори *Intel 486DX2* та *Fujitsu SPARClite '934*

<i>Intel 486DX2</i>				<i>Fujitsu SPARClite '934</i>			
Інструкція	Струм (mA)	Цикли	Енергія (10 ⁻⁸ J)	Інструкція	Струм (mA)	Цикли	Енергія (10 ⁻⁸ J)
nop	276	1	2.27	nop	198	1	3.26
mov dx,[bx]	428	1	3.53	ld [10],i0	213	1	3.51
mov dx,bx	302	1	2.49	or g0,i0,10	198	1	3.26
mov [bx],dx	522	1	4.30	st i0,[10]	346	2	11.4
add dx,bx	314	1	2.59	add i0,o0,10	199	1	3.28
add dx,[bx]	400	2	6.60	mul g0,r29,r27	198	1	3.26
jmp	373	3	9.23	srl i0,1,10	197	1	3.25

Як тільки побудована енергетична модель за інструкцією для конкретного процесора, загальна вартість енергії, E_p , будь-якої даної програми, P , визначається як

$$E_p = \sum_i (Base_i * N_i) + \sum_{I,j} (Inter_{i,j} * N_{i,j}) + \sum_k E_k,$$

де $Base_i$ - базова вартість інструкцій;

N_i - кількість виконань інструкції I ;

$Inter_{i,j}$ - накладні витрати на потужність між інструкціями, коли за інструкцією i слідує інструкція j ;

$N_{i,j}$ - кількість разів, коли пара (i, j) виконується;

E_k - затрати енергії пов'язані з перемиканням між інструкціями, втратами в виконанні конвеєра і помилками кешу.

1.2. Повні енергетичні моделі на системному рівні

Мало користі від вивчення та оптимізації лише ядра центрального процесора, якщо інші компоненти мають значний вплив на споживання енергії або навіть домінують на ньому. Тому необхідно врахувати інші критичні компоненти, щоб зменшити загальну енергію системи. У підрозділі 1.2.1 розглядаються апаратні моделі на рівні стану, де загальне споживання енергії всієї системи оцінюється на основі стану кожного пристрою, що перебуває або переходить у/з. Тут передбачається, що кожен пристрій здатний переходити в один із декількох енергозберігаючих станів, таких як режим сну, залежно від потреби на цьому конкретному пристрої [13]. Ця можливість зазвичай надається в портативних системах, щоб продовжити своє життя якомога довше. Представлені програмні підходи у підрозділі 1.2.2 визначають точки доступу до енергії в додатках та процедурах операційної системи, і таким чином дозволяють програмістам видаляти вузькі місця або модифікувати програмне забезпечення таким чином, щоб воно було енергоефективним. Нарешті, у підрозділі 1.2.3 представлений повний інструмент моделювання на системному рівні, який моделює апаратні компоненти, такі як ЦП,

ієрархія пам'яті та підсистема диска з низьким енергоспоживанням, а також програмні компоненти, такі як ОС та додатки.

1.2.1. Апаратна енергетична модель стану

Сігнетті та ін. представив загальносистемну методику оптимізації енергії з апаратною енергетичною моделлю, заснованою на стані [13]. Ця модель живлення інкапсулює деталі низького рівня кожної апаратної підсистеми, визначаючи набір станів живлення (наприклад, сплячий режим, дрімання або зайнятість для ЦП) для кожного пристрою. Кожен стан живлення характеризується енергоспоживанням апаратних засобів під час стану, який називається стаціонарним живленням. Крім того, кожному переходу між станами призначається витрата енергії, яка називається перехідною енергією. Оскільки переходи між станами відбуваються в результаті системних викликів, відповідну енергію можна виміряти, відстежуючи системні дзвінки.

Вищезгадана енергетична модель стану була реалізована як розширення для емулятора *Palm OS (POSE)* [80], що є додатком на базі *Windows*, що імітує функції пристрою *Palm*. *POSE* емулює *Palm OS* та виконання інструкцій мікропроцесора *Motorola Dragonball* [79]. Для кількісної оцінки енергоспоживання пристрою та надання параметрів симулятору були проведені вимірювання з метою фіксації перехідного споживання енергії, а також постійного споживання енергії, як представлено в табл. 1.4 та 1.5 [13]. Пристрій *Palm* від *IBM* було підключено до джерела живлення за допомогою осцилографа, що вимірює напругу на невеликому резисторі. Споживання енергії основних апаратних підсистем, таких як центральний процесор, рідкокристалічний дисплей, підсвічування, кнопки, ручка та послідовний зв'язок, вимірювали за допомогою вимірювальних програм під назвою *Power* та *Millywatt* [40].

Таблиця 1.4

Стабільний стан пристрою *Palm* від *IBM*

Пристрій	Стан	Потужність (мВт)
ЦП	Зайнятий	104.502
	Бездіяльність	0.0
	Сон	-44.477
<i>LCD</i>	Ввімкнений	0.0
	Вимкнений	-20.961
Підсвічування	Ввімкнений	94.262
	Вимкнений	0.0
Кнопка	Натиснута	45.796
Курсор	На екрані	82.952
	Графіті	86.029

Таблиця 1.5

Перехідна потужність пристрою *Palm* від *IBM*

Системна команда	Перехідна потужність (мДж)
ЦП сон	2.025
ЦП пробудження	11.170
<i>LCD</i> сон	11.727
Ключовий сон	2.974
Курсор відкритий	1.935

1.2.2. Енергетична модель на основі процесів

Оскільки програмне забезпечення є головним фактором, що визначає діяльність апаратних компонентів, таких як ядро процесора, система пам'яті та шини, існує потреба у дослідженні енергоорієнтованих програм програмного забезпечення та їх взаємодії та

інтеграції з продуктивним програмним забезпеченням. У цьому підрозділі представлені методи вимірювання потужності на основі процесів для оптимізації системи [12, 20]. Використовуючи спеціально розроблені інструменти моніторингу, ці методи, що базуються на вимірах, орієнтуються на споживання енергії всієї системи та намагаються вказати на гарячі точки в додатках та процедурах операційної системи.

У *PowerScope* [16] статистичний дискретизатор, керований часом, використовується для того, щоб визначити, яка частка загальної енергії споживається протягом певного періоду часу внаслідок специфічних процесів у системі. Ця методика може бути додатково розширена для визначення енергоспоживання різних процедур у процесі. Забезпечуючи такий детальний зворотний зв'язок, *PowerScope* допомагає зосередитися на ті компоненти системи, які відповідають за основну частину споживання енергії. Чанг та ін. передбачав подібний інструмент, але він базується на статистичній вибірці на основі енергії, яка використовує споживання енергії для стимулювання збору зразків [9]. Мультиметр [16] (або лічильник енергії [9]) контролює споживання енергії системою та програмним забезпеченням, що тестується, генеруючи переривання для кожного інтервалу часу [16] (або кожного кванту енергії [9]). Це переривання запропонує системі записати ідентифікатор процесу поточно запущеного процесу, а також зібрати поточний [16] (або енергетичний [9]) зразок. Після експерименту зібрані дані, тобто ідентифікатори процесів та зразок струму / енергії, аналізуються в автономному режимі для узгодження процесів із зразками енергії для створення енергетичного профілю.

Результати цього дослідження показали, що операційна система витрачала нетривіальну кількість енергії порівняно з іншими процесами користувача. Крім того, часто існують суттєві відмінності між керованими часом та енергетичними профілями, і тому необхідно ретельно поєднувати обидва методи вибірки для отримання більш точної інформації про енергетичний профіль.

1.2.3. Специфічна для компонентів енергетична модель

Згадані вище методи профілювання електроенергії забезпечують енергетичні витрати на виконання певної програми, але без розуміння загальної поведінки системи достатньо детально, щоб охопити взаємодію між усіма компонентами системи. Повний системний симулятор живлення *SoftWatt* [23] вирішує цю проблему шляхом моделювання апаратних компонентів, таких як процесор, ієрархія пам'яті та підсистема диска, а також кількісно визначає поведінку живлення як прикладного програмного забезпечення, так і операційної системи. *SoftWatt* був побудований поверх інфраструктури *SimOS* [82], яка забезпечує детальне моделювання як апаратного, так і програмного забезпечення, включаючи операційну систему IRIX [77]. Для того, щоб зафіксувати повну поведінку системи в системі, *SoftWatt* інтегрує різні аналітичні моделі потужності, доступні в інших дослідженнях, в різні апаратні компоненти *SimOS*.

Досвід роботи програмного пакету *JVM98* з програмного забезпечення *JVM98* [84] від *SPEC* [85] наголосив на важливості повного моделювання системи для аналізу впливу як архітектури, так і ОС на виконання додатків. З точки зору системного обладнання, диск є найбільшим споживачем електроенергії всієї системи. Однак із прийняттям малопотужного диска точка доступу була переведена до мережі розподілу та генерації тактової частоти процесора. Також виявлено, що підсистема кешу споживає більше енергії, ніж ядро процесора. З точки зору програмного забезпечення, користувальницький режим споживає більше енергії, ніж режим ядра. Однак певні служби ядра називаються так часто, що на них припадає значне споживання енергії в ієрархії процесора та пам'яті. Таким чином, врахування енергоспоживання коду ядра є критичним для зменшення загальних енергетичних витрат. Нарешті, перехід підсистеми центрального процесора та пам'яті в режим низького енергоспоживання або навіть зупинка процесора під час виконання простою може значно зменшити споживання енергії.

1.3. Моделі енергії на рівні взаємозв'язку в паралельних системах

Після представлення енергетичних моделей на рівні процесора (розділ 1.1) та системному рівні (розділ 1.2), цей підрозділ описує енергетичні моделі на паралельному системному рівні з акцентом на мережах взаємозв'язку. З постійно зростаючим попитом на обчислювальну потужність процесори стають дедалі більш взаємопов'язаними, створюючи великі кластери комп'ютерів, що обмінюються даними мереж, що під'єднані до інших мереж. Ван та співавтори показали, що енергоспоживання цих компонентів зв'язку стає все більш критичним, особливо зі збільшенням пропускної здатності мережі та пропускної здатності до гігабітних та терабітних доменів [53]. Таким чином, аналіз потужності в цій області зазвичай націлений на будівельні блоки всередині мережевого маршрутизатора та комутаційної тканини.

Бітова енергетична модель [57] розглядає енергію, споживану для кожного біта, що рухається всередині комутаційної обчислювальної одиниці від вхідного до вихідного портів, як підсумок бітової енергії, що споживається на кожному з наступних трьох компонентів: внутрішні комутатори вузлів, які направляють пакет з одного проміжного етапу до наступного, поки не дійде до порту призначення; внутрішні черги буферів, які зберігають пакети з нижчими пріоритетами, коли виникає суперечка; та з'єднувальні дроти, які розсіюють потужність, коли біт, що передається на дроті, перевертає полярність з попереднього біта. Для кожного з цих компонентів використовувались різні моделі на основі їхніх характеристик. Наприклад, розрядна енергія перемикача вузлів залежить від стану; це залежить від наявності або відсутності пакетів на інших вхідних портах. З іншого боку, енергоспоживання внутрішнього буфера може бути виражене як сума енергії доступу до даних (зчитування та запис) та операції оновлення пам'яті. Нарешті, розрядна енергія з'єднувальних проводів залежить від ємності дроту, довжини та зв'язку між сусідніми проводами.

На відміну від підходу на рівні бітів, згаданого вище, в [53] був представлений симулятор енергоефективності мережевого рівня на рівні архітектури. *Orion* моделює

мережу взаємозв'язку, що складається з генерації повідомлень (таких як джерела), транспортування (буфери маршрутизатора, поперечні лінії, арбітри та компоненти зв'язку) та споживання (поглиначів) агентів. Кожен із цих агентів є будівельним блоком мережі взаємозв'язку і представлений моделлю енергетичної моделі на рівні архітектури. Ця енергетична модель базується на ємності комутатора кожного компонента, включаючи ємності засувки та дроту. Ці рівняння ємності поєднуються з оцінкою активності комутації для обчислення споживання енергії на роботу компонента. *Orion* можна використовувати для підключення і відтворення маршрутизатора та компонентів зв'язку для формування різних архітектур мережевої тканини.

Висновки до розділу

Споживана центральним процесором енергія є основною частиною загального енергоспоживання комп'ютерної системи і, отже, стала основною метою аналізу енергоспоживання. Кілька моделей живлення були розроблені та інтегровані в існуючі симулятори продуктивності, щоб дослідити споживання енергії центрального процесора як на основі функціональних блоків, так і на процесорі в цілому.

Однак необхідно було врахувати й інші критичні компоненти, щоб зменшити загальну енергію системи. Були розглянуті апаратні моделі на рівні стану, де загальне споживання енергії всієї системи оцінюється на основі стану кожного пристрою, що допомагає продовжити життя системи якомога довше. Представлені програмні підходи, що визначають точки доступу до енергії в додатках та процедурах операційної системи, і таким чином дозволяють програмістам видаляти вузькі місця або модифікувати програмне забезпечення таким чином, щоб воно було енергоефективним. Представлений повний інструмент моделювання на системному рівні, який моделює апаратні компоненти, такі як ЦП, ієрархія пам'яті та підсистема диска з низьким енергоспоживанням, а також програмні компоненти, такі як ОС та додатки.

Після представлення енергетичних моделей на рівні процесора та системному рівні, були описані енергетичні моделі на паралельному системному рівні з акцентом на мережах взаємозв'язку. З постійно зростаючим попитом на обчислювальну потужність процесори стають дедалі більш взаємопов'язаними, створюючи великі кластери комп'ютерів, що обмінюються даними мереж, що під'єднані до інших мереж. Аналіз потужності в цій області націлений на будівельні блоки всередині мережевого маршрутизатора та комутаційної тканини.

РОЗДІЛ 2

МЕТОДИ ДИНАМІЧНОГО УПРАВЛІННЯ ЖИВЛЕННЯМ (*DPM*)

Хоча методи моделювання та вимірювання, описані в розділі 1, спрямовані на оптимізацію показників потужності під час проектування, методи *DPM* націлені на зменшення споживання енергії під час роботи шляхом вибіркового вимкнення або уповільнення компонентів, коли системи простоюють або обслуговують невеликі робочі навантаження. Як і в розділі 1, методи *DPM* застосовуються по-різному та на різних рівнях. Наприклад, техніка динамічного масштабування напруги (*DVS*) працює на рівні центрального процесора і змінює напругу живлення та робочу частоту процесора під час роботи як метод управління живленням [54]. Подібна техніка, яка називається *DLS*, працює на рівні взаємозв'язку і переводить комутаційні комутатори в кластерній системі в режим низької енергії для економії енергії [27]. Методи *DPM* також можуть бути використані для вимкнення пристроїв вводу-виводу в режимі очікування [39],

Як узагальнено в таблиці 2.1, у цьому розділі розглядаються методи *DPM*, класифіковані на основі рівня впровадження. У розділі 2.1 розглядаються методи *DPM*, що застосовуються на рівні ЦП. У розділі 2.2 обговорюються підходи *DPM* на системному рівні, що розглядають інші системні компоненти (пам'ять, жорсткий диск, пристрої вводу-виводу, дисплей тощо), ніж ЦП. Нарешті, у розділі 2.3 представлені методи *DPM*, запропоновані для паралельних систем, де кілька вузлів співпрацюють, щоб заощадити загальну потужність, колективно виконуючи задане паралельне завдання.

Таблиця 2.1

Таксономія динамічних методів управління енергією

Тип	Рівень реалізації	Механізм моніторингу	Механізм управління	Розділ
1	2	3	4	5
ЦП	Рівень ЦП	Відстежуйте активність внутрішньої шини до зменшити активність комутації	Різні схеми кодування [29, 61, 69], на основі компілятора планування [31, 63, 66]	2.1.1
		Моніторинг інструкцій процесора у виконанні для управління подачею годинника до кожного компоненту	<i>Clock gating</i> для компонентів процесора [21, 26]	2.1.2
		Відстежуйте робоче навантаження процесора для регулювання напруга живлення процесора	<i>DVS</i> з інтервальним або планувальником на основі історії [25, 52, 58, 68], планувальник на основі компілятора [4, 5, 22, 54]	2.1.3
Система	На базі АЗ	Відстежуйте активність пристрою для закриття його або сповільнення	Час очікування, прогнозовані або стохастичні політики [7, 8, 14, 15, 17, 24, 28, 51, 57, 60]	2.2.1
	На базі ПЗ	Моніторинг активності пристрою за допомогою прикладного або системного програмного забезпечення для закриття його або сповільнення	Прогнозування майбутнього використання пристрою [24, 35, 36, 38], <i>ACPI</i> [1, 2, 47]	2.2.2

Продовження таблиці 2.1

1	2	3	4	5
Паралельна система	На базі АЗ	Відстежуйте кілька робочих навантажень центрального процесора для спільного регулювання напруги живлення	<i>CVS</i> (Злагоджена <i>DVS</i>) [15]	2.3.1
		Моніторинг активності комутатора / маршрутизатора до перебудови підключення або поставлення в режим зниженої потужності	<i>DVS</i> на основі історії комутатор / маршрутизатор [43], , <i>Dynamic Link Shutdown</i> [27]	2.3.1
	На базі ПЗ	Відстежуйте синхронізацію, щоб вимкнути обертання вузлів	Економний бар'єр [28]	2.3.2
		Моніторинг розподілу навантаження при відключенні деяких вузлів	Розбалансування навантаження [40]	2.3.3

2.1. DPM на рівні процесора

Інтуїція економії енергії на рівні центрального процесора походить від основних характеристик енергоспоживання цифрових статичних схем *CMOS*, що дається

$$E \propto C_{eff} V^2 f_{CLK}, \quad (2.1)$$

де C_{eff} - ефективна комутаційна ємність операції;

V - напруга живлення;

f_{CLK} - тактова частота [21].

Методи *DPM*, представлені в цьому розділі, зменшують споживання енергії, орієнтуючись на один або кілька з цих параметрів. У підрозділі 2.1.1 обговорюються методи зменшення активності комутації процесора, головним чином на шляху передачі даних та шин. У підрозділі 2.1.2 обговорюються прийоми синхронізації годин, які зменшують споживання енергії, вимикаючи годинник простою компонента, тобто $f_{CLK} = 0$. Зрештою, підрозділ 2.1.3 представляє одну з найбільш перспективних а також найскладніша методика *DPM* на рівні процесора, заснована на *DVS*. *DVS* масштабує як V , так і f_{CLK} для обслуговування робочого навантаження процесора з мінімальною необхідною потужністю. Якщо точніше, *DVS* забезпечує значну економію енергії, не впливаючи на продуктивність.

2.1.1. Зменшення активності комутації

Як обговорювалося раніше, зменшення комутаційної активності відіграє важливу роль у зменшенні споживання енергії. Ряд таких методів оптимізації було запропоновано для зменшення активності комутації внутрішніх шин [29, 61, 69] та функціональних блоків [31, 63, 66] процесора. У випадку з шинами енергія споживається, коли дроти змінюють стан (від 0 до 1). Для зменшення активності комутації на шинах використовуються різні техніки за рахунок зменшення кількості переходів дроту. Стен і Бурлесон запропонували кодування з використанням шини-інверту, де значення шини інвертується, коли більше половини проводів змінюють стан [48]. Іншими словами, коли нове значення, яке передається по шині, відрізняється більш ніж на половину своїх бітів від попереднього значення, тоді всі біти інвертуються перед передачею. Це зменшує кількість змін стану на дроті і тим самим економить енергію.

Хенкель та Лекацас запропонували більш складний підхід, коли таблиці кеш-пам'яті використовуються на стороні передачі та прийому каналу для подальшого зменшення переходів [25]. Тобто, коли на вході каналу спостерігається значення “hit”, система буде відправляти лише індекс запису кешу замість цілого значення даних, що

зменшить кількість переходів. Нарешті, Вен та інші використовували шифрувальне кодування для зменшення шинного трафіку і, отже, потужності на основі стиснення даних на дротах шини [55]. Як вдосконалення цієї техніки було запропоновано також перехідне кодування, де кодування даних являє собою дотові зміни, а не абсолютне значення, що спрощує проблему оптимізації енергії.

З іншого боку, активність перемикання процесора також може бути зменшена за допомогою прийомів компілятора, що усвідомлює енергію. Хоча вони застосовуються під час компіляції, вони розглядаються як методи *DPM*, оскільки їх ефект тісно пов'язаний із поведінкою системи під час виконання. Наприклад, у плануванні інструкцій техніки [63, 66], інструкції переупорядковуються, щоб зменшити активність перемикання між послідовними інструкціями. Більш конкретно, це мінімізує активність перемикання шини даних між кеш-пам'яттю мікросхеми та основною пам'яттю при пропуску кешу інструкцій [52]. Холодне планування [49] надає пріоритет вибору наступної інструкції для виконання на основі енергетичних витрат на розміщення цієї інструкції в розкладі. Інша техніка, заснована на компіляторі, яка називається присвоєнням регістру [26], зосереджена на зменшенні активності перемикання на шині шляхом перемаркування полів реєстру згенерованих компілятором інструкцій. Симулятор, такий як *SimplePower* [56], використовується для параметризації компілятора з використанням зразків слідів. Іншими словами, він фіксує перехід частоти між мітками регістрів в інструкціях, що виконуються в послідовних циклах, і ця інформація потім використовується для отримання кращого кодування для регістрів, таким чином, що активність перемикання і, отже, споживання енергії на шині зменшується.

2.1.2. *Clock gating*

Clock gating передбачає заморожування годинника простою компонента. Енергія економиться, оскільки в цих заморожених одиницях не буде передаватися сигнал чи

дані. Широке використання годинників широко застосовується, оскільки це концептуально просто; годинник можна перезапустити, просто знявши з положення сигнал заморожування годинника. Тому необхідні лише невеликі накладні витрати з точки зору додаткової схеми, і компонент може переходити з режиму очікування в активний стан лише за кілька циклів. Ця техніка була впроваджена в декількох комерційних процесорах, таких як *Alpha 21264* [22] та *PowerPC 603* [17]. *Alpha 21264* використовує ієрархічну архітектуру синхронізації із закритими годинниками. Залежно від інструкції, яку потрібно виконати, кожна одиниця центрального процесора (наприклад, одиниця з плаваючою точкою) здатна заморозити годинник до своїх підкомпонентів (наприклад, суматор, дільник і множник в одиниці з плаваючою точкою).

Процесор *PowerPC 603* підтримує кілька режимів сну, заснованих на синхронізації годинника. Для цього він має два типи контролерів годин: глобальний та локальний. Годинники для деяких компонентів регулюються на глобальному рівні, а інші - на місцевому. Наприклад, розглянемо *PLL*, який діє головним чином як стабілізатор частоти і не залежить від глобального тактового сигналу.

Незважаючи на те, що годинники для всіх блоків глобально відключені, а процесор перебуває в режимі сну, *PLL* може продовжувати функціонувати, що робить можливим швидке пробудження (протягом десяти тактових циклів). З іншого боку, якщо *PLL* також вимкнено, буде досягнуто максимальної економії енергії, але час пробудження може становити до 100 с, щоб *PLL* зміг переблокуватися на зовнішній годинник.

2.1.3. Динамічне масштабування напруги (*DVS*)

На відміну від синхронізації годинника, яку можна застосувати лише до непрацюючих компонентів, *DVS* націлений на компоненти, які перебувають у активному стані, але обслуговують невелике робоче навантаження. Це було запропоновано як засіб

для процесора для забезпечення високої продуктивності, коли це потрібно, при цьому суттєво зменшуючи споживання енергії під час низьких робочих навантажень. Перевагу *DVS* можна спостерігати з характеристик енергоспоживання цифрових статичних схем *CMOS* та рівняння тактової частоти

$$delay \propto \frac{V}{(V-V_k)^\alpha} \text{ та } f_{CLK} \propto \frac{(V-V_k)^\alpha}{V}, \quad (2.2)$$

де V - напруга живлення;

f_{CLK} - тактова частота;

α - коливається від 1 до 2;

V_k - залежить від порогової напруги, при якій насичення швидкості і відбувається [21].

Зменшення напруги джерела живлення призведе до квадратичного зменшення споживання енергії, як показано у формулі (2.1). Однак це може створити більшу затримку розповсюдження і одночасно змусити зменшити тактову частоту, як показано у формулі (2.2). Хоча зазвичай бажано встановити якнайвищу частоту для більш швидкого виконання інструкцій, частоту тактової частоти та напругу живлення можна зменшити для деяких завдань, де не потрібна максимальна швидкість виконання. Оскільки активність процесора є змінною, існують періоди простою, коли не виконується жодна корисна робота, і *DVS* може бути використаний для усунення цих витрат енергії, що витрачаються в режимі очікування, знижуючи напругу та частоту процесора.

Насичення швидкості пов'язане з порогом напівпровідникової напруги, після якого відбувається насичення і поведінка транзистора стає нелінійною.

Для того, щоб наочно продемонструвати переваги методів *DVS*, на рис. 2.1 порівнюється *DVS* із простою схемою увімкнення / вимкнення, де процесор просто вимикається, коли він не працює (протягом часу 2 ~ 4, 5 ~ 7 та 8,5 ~ 11 (рис 2.1)). *DVS* зменшує напругу та частоту, поширюючи навантаження на більш тривалий період, але більш ніж квадратично зменшуючи споживання енергії. Швидкий розрахунок з рис. 2.1

показує приблизно 82% зменшення потужності на основі рівняння (2.1), оскільки $E_{DVS}/E_{On/Off} = (4*(0,5)^3 + 3*(0,33)^3 + 4*(0,375)^3) / (2*1^3 + 1*1^3 + (1,5)*1^3) = 0,82 / 4,5 = 0,18$.

Примітка що кожне робоче навантаження завдання, яке представлено площею всередині прямокутника на рис. 2.1., залишається однаковим як для простих механізмів увімкнення / вимкнення, так і для *DVS*.

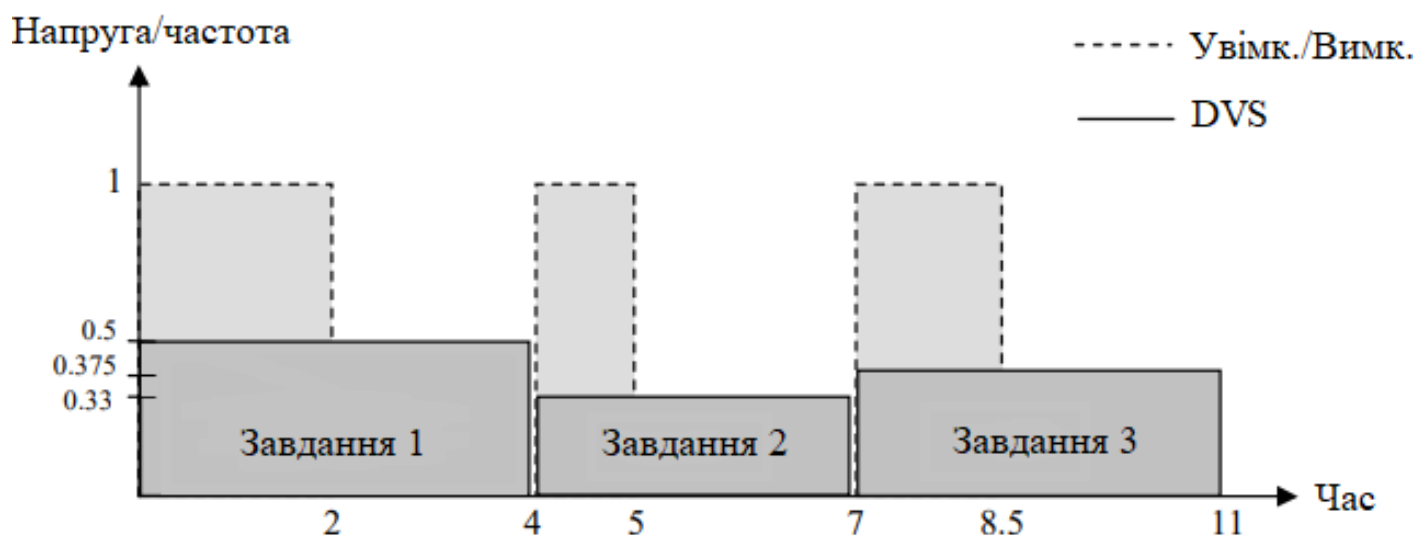


Рис. 2.1. Графік планування напруги з механізмами увімкнення/вимкнення та *DVS*

Поточні спеціальні та комерційні чіпи *CMOS* здатні надійно працювати при різних напругах живлення [46, 64], і існує ряд комерційно доступних процесорів, які підтримують механізми *DVS*. У табл. 2.2 наведено мобільний процесор *Intel Pentium III* з 11 рівнями частоти та 6 рівнями напруги з двома режимами роботи: режим максимальної продуктивності та оптимізований режим роботи батареї [34]. Режим максимальної продуктивності розроблений для забезпечення найкращої продуктивності, тоді як оптимізований режим роботи батареї забезпечує баланс між продуктивністю та часом роботи акумулятора. Процесор *Crusoe* від *Transmeta, Inc.* також має різні напруги та частоти, як представлено в табл. 2.3.

Таблиця 2.2

Тактова частота проти напруги живлення для мобільного процесора *Intel Pentium III*

Режим максимальної продуктивності			Оптимізований режим роботи акумулятора		
Частота (МГц)	Напруга (В)	Макс. потужність споживання (Вт)	Частота (МГц)	Напруга (В)	Макс. потужність споживання (Вт)
500	1.10	8.1	300	0.975	4.5
600	1.10	9.7	300	0.975	4.5
600	1.35	14.4	500	1.10	8.1
600	1.60	20.0	500	1.35	12.2
650	1.60	21.5	500	1.35	12.2
700	1.60	23.0	550	1.35	13.2
750	1.35	17.2	500	1.10	8.1
750	1.60	24.6	550	1.35	13.2
800	1.60	25.9	650	1.35	15.1
850	1.60	27.5	700	1.35	16.1
900	1.70	30.7	700	1.35	16.1
1000	1.70	34.0	700	1.35	16.1

Таблиця 2.3

Тактова частота проти напруги живлення для процесора *Transmeta Crusoe*

Частота (МГц)	Напруга (В)	Споживання енергії (Вт)
667	1.6	5.3
600	1.5	4.2
533	1.35	3.0
400	1.225	1.9
300	1.2	1.3

Основна проблема при застосуванні *DVS* - це знати, коли і як масштабувати напругу та частоту. У наступному обговоренні представлені три різні планувальники напруги: інтервальний, міжзадачний та внутрішньозадачний планувальник.

Інтервальний планувальник - це часовий планувальник напруги, який передбачає майбутнє навантаження, використовуючи історію робочого навантаження. Планувальники міжзадач та внутрішніх задач націлюються на програми в режимі реального часу з термінами виконання завдань. Планувальник міжзадач змінює швидкість на кожній межі завдання, тоді як внутрішньозадачний планувальник змінює швидкість в межах одного завдання за допомогою компіляторів. Підходи між завданнями використовують попередні знання програми для створення прогнозів для даного завдання, тоді як підходи до завдань намагаються врахувати виграш часу слабкості, який виникає внаслідок різниці у шляху виконання програми, спричиненої умовними операторами.

2.1.3.1. Планувальник на основі інтервалів

На основі інтервалу планувальники напруги [25, 68] ділять час на однакові інтервали довжини та аналізують використання процесора попередніх інтервалів, щоб визначити напругу / частоту наступного інтервалу. Говіл та ін. обговорили та порівняли сім таких алгоритмів [21]:

- 1) *PAST* використовує недавнє минуле як провісник майбутнього.
- 2) *FLAT* просто намагається згладити швидкість процесора до глобальних середніх значень.
- 3) *LONG_SHORT* намагається знайти золоту середину між останньою поведінкою та середньою тривалістю.
- 4) *AGED_AVERAGES* використовує метод експоненціального згладжування, намагаючись передбачити за допомогою середньозваженого.
- 5) *CYCLE* - це більш складний алгоритм прогнозування, який намагається скористатися попередніми значеннями *run_percent*, що мають циклічну поведінку, де *run_percent* - це частка циклів в інтервалі, протягом якого центральний процесор активний.

- 6) *PATTERN* - це узагальнена форма ЦИКЛУ, яка намагається ідентифікувати останні значення *run_percent* як повторюваний шаблон.
- 7) *PEAK* є більш спеціалізованою версією *PATTERN* і використовує наступну евристику, засновану на спостереженні на вузьких піках: збільшення *run_percents* буде падати, але зменшення *run_percents* буде продовжувати падати [21].

Згідно з їх імітаційними дослідженнями, прості алгоритми, засновані на раціональному згладжуванні, а не на складних схемах прогнозування, показали кращу продуктивність. Їх дослідження також показують, що існують подальші можливості шляхом вдосконалення прогнозів, таких як сортування минулої інформації за типом process або надання корисної інформації програмами [21].

2.1.3.2. Методи міжзадачних завдань для додатків у режимі реального часу

Планувальник, заснований на інтервалах, простий і легкий у реалізації, але часто неправильно передбачає майбутні робочі навантаження та погіршує якість обслуговування. У додатках, що не працюють у режимі реального часу, незавершене завдання з попереднього інтервалу буде виконано пізніше і не спричинить серйозних проблем. Однак у додатках у режимі реального часу завдання визначаються часом початку завдання, необхідними обчислювальними ресурсами та терміном виконання завдання. Отже, масштабування напруги / частоти повинно проводитися з обмеженням, щоб не пропускати терміни. Оптимальний графік визначається таким, для якого всі завдання виконуються під час або до закінчення строків, а загальна спожита енергія мінімізується.

Для набору завдань із заданими часовими параметрами, такими як терміни, побудова оптимального графіка напруги вимагає надлінійної алгоритмічної складності. Одним простим евристичним алгоритмом є ідентифікація завдання з найраннішим кінцевим терміном та знаходження мінімальної постійної швидкості, необхідної для виконання завдання протягом часового інтервалу перед кінцевим терміном. Повторення

однієї і тієї ж процедури для всіх завдань забезпечує графік напруги. Куан та Фу запропонували більш ефективний алгоритм планування між завданнями для додатків у реальному часі [42]. Цей підхід намагається знайти критичні інтервали з використанням заданих часових параметрів, таких як час початку та кінцеві терміни, які можуть бути вузьким місцем при виконанні набору завдань. Потім складається графік напруги для набору критичних інтервалів, і повний графік напруги з низькою енергією будується на основі мінімальної постійної швидкості, знайденої протягом будь-якого критичного інтервалу. Хоча цей жадібний підхід гарантує мінімальне споживання енергії, він не завжди може забезпечити графік мінімальної енергії.

Інша техніка *DVS* між завданнями була запропонована для конкретного додатку в режимі реального часу, плеєра *MPEG* [13, 58]. Завданням цього є декодування кадру *MPEG* або групи зображень (*GOP*).

Оскільки різні кадри вимагають порядку різних обчислювальних накладних витрат для декодування, вигідніше змінювати напругу живлення та робочу частоту залежно від кадрів, а не *GOP*. Основна складність полягає в передбаченні наступного навантаження (наприклад, декодування наступного кадру) для того, щоб призначити належну настройку напруги та частоти. Якщо наступне робоче навантаження (час декодування коду) недооцінено, буде призначено напругу/частоту, що нижча за необхідну, і завдання не досягне свого кінцевого терміну, що спричинить втрату тремтіння або кадрів, а якість відео погіршиться. З іншого боку, якщо наступне навантаження переоцінено, буде встановлено напругу/частоту, яка перевищує необхідну, що призведе до більшого споживання енергії, ніж потрібно.

Сон та ін. запропонували два евристичні алгоритми *DVS* для декодування *MPEG* [46]: *DVS-DM* (*DVS* з алгоритмом мінімізації затримки та швидкості падіння) та *DVS-PD* (*DVS* з передбаченням часу декодування). *DVS-DM* - це інтервал на основі *DVS* в тому сенсі, що він планує напругу на кожній межі *GOP* на основі параметрів (переважно затримки та швидкості падіння), отриманих з попередньої історії декодування. *DVS-PD* визначає напругу на основі інформації з наступного *GOP* (наприклад, розміри кадрів та

типи кадрів), а також попередньої історії. Оскільки кадри мають різні характеристики залежно від типу кадру, *DVS-PD* пропонує вищу точність прогнозування для майбутнього робочого навантаження порівняно з *DVS-DM* [46].

Чедід запропонував інший набір методів для енергозберігаючого декодування *MPEG* [10]: регресія, діапазон-середній та діапазон-максимальний. Методика регресії заснована на спостереженні, що розподіл розміру кадру/часу декодування слідує лінійній регресійній моделі [3] з високою точністю, як показано на рисунку 2.2. Лінія регресії будується динамічно під час виконання, обчислюючи нахил співвідношення розмір кадру/часу декодування на основі минулої історії. Інші дві техніки, *range-avg* і *range-max*, полегшують обчислювальні накладні витрати, знайдені в алгоритмі регресії. Ці підходи поділяють розподіл часу декодування/розміру кадру на кілька діапазонів, як на рис. 2.2, і приймають рішення щодо оцінки на основі середнього часу декодування або максимального часу декодування у кожному діапазоні.

Таблиця 2.4 узагальнює різні міжзадачні методи *DVS* для декодування *MPEG*, про які йшлося в попередніх параграфах.

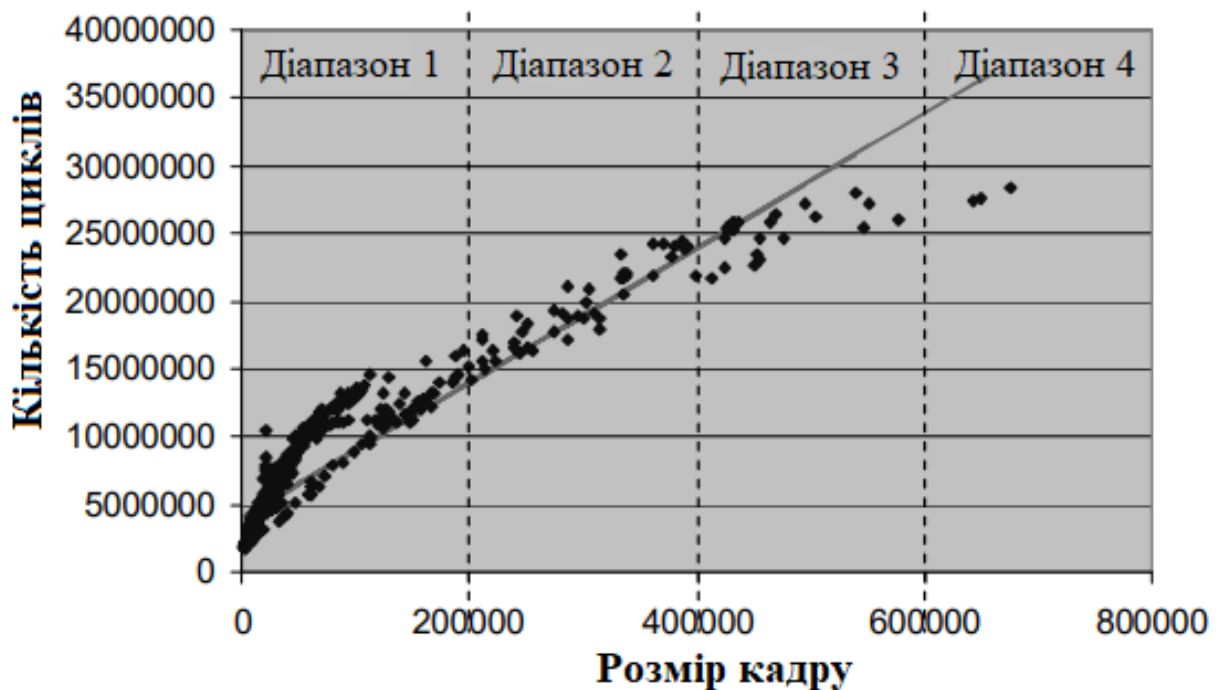


Рис. 2.2. Час декодування у залежності від розміру кадру (за мотивами фільму "Недотримання")

Таблиця 2.4

Методи *DVS* між завданнями для програми в режимі реального часу (програвач *MPEG*)

Техніка	Рівень впровадження	Метод, який використовується для прогнозування майбутнього навантаження	Переваги	Недоліки
<i>DVS-DM</i>	<i>GOP</i> (група фотографій)	Попередня історія затримки та падіння	Легко реалізувати	Неточний, якщо робоче навантаження коливається
<i>DVS-PD</i>	<i>GOP</i> (група фотографій)	Середньозважена попередня історія та наступна інформація про <i>GOP</i>	Точніший і менш вразливий до коливань, ніж <i>DVS-DM</i>	Вразливий до коливань в межах кожного <i>GOP</i>
Регресія	Фоторамка	Динамічна регресія попередньої історії та інформації про наступний кадр	Високоточний прогноз	Обчислювально дорогий
<i>Range-avg</i>	Фоторамка	Середнє навантаження минулих фоторамок з подібною рамкою, типом і розміром	Простий у реалізації та гнучкий в балансуванні між енергозбереженням та якістю відео	Менш точний, ніж регресія
<i>Range-max</i>	Фоторамка	Максимальне навантаження минулих фоторамок з подібною рамкою, типом і розміром	Простий у реалізації та більш гнучка, ніж <i>Range-avg</i>	Менш точний, ніж регресія та <i>Range-avg</i>

2.1.3.3. Прийоми внутрішньозадачних задач для додатків у режимі реального часу

На відміну від згаданих вище методів *DVS* між завданнями, де зміни напруги/частоти відбуваються між послідовними завданнями, техніки *DVS* всередині завдання застосовуються під час виконання завдання за допомогою потужного компілятора. Компілятор визначає різні можливі варіанти виконання шляхів в межах завдання, кожне з яких вимагає різного обсягу роботи і, отже, різного налаштування напруги/частоти. Розглянемо приклад завдання в реальному часі та його графік потоку на рис. 2.3. . На рис. 2.3(b) кожен вузол представляє основний блок, B_i , цього завдання та номер у кожному вузлу позначає кількість циклів виконання, необхідних для завершення блоку. Загальна кількість циклів змінюється для одного і того ж завдання в залежності від обраного шляху, а результуючий проміжок часу є метою оптимізації в наступних техніках внутрішнього завдання.

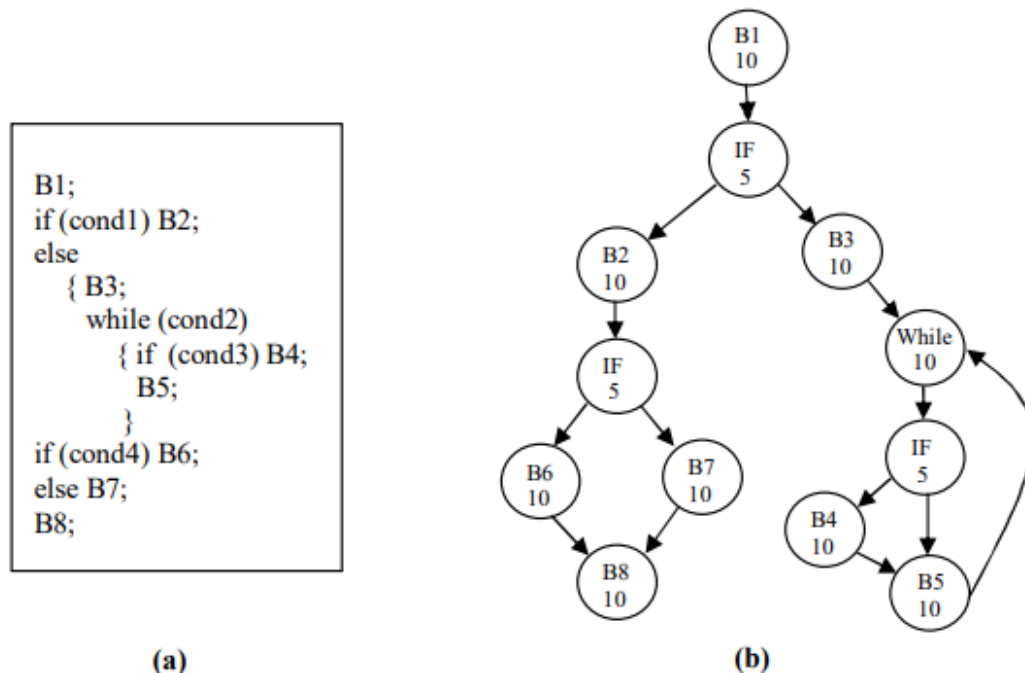


Рис.2.3. Шляхи внутрішнього завдання: (a) приклад програми та (b) її графік потоку

Азеведо та ін. запровадив внутрішньозадачну техніку *DVS* з використанням контрольних точок програми під контролем компілятора [2]. Контрольні точки вказують місця в програмі, де напруга / частота процесора повинна бути перерахована та

масштабована. Програма профілюється, використовуючи репрезентативний набір вхідних даних, і збирається інформація про мінімальну / максимальну витрачану енергію та кількість циклів між пунктами пропуску. Ця інформація використовується в планувальнику напруги під час виконання, щоб регулювати напругу енергоефективним способом, одночасно дотримуючись терміну.

Подібним чином Шін і Кім запропонували інструмент перетворення на основі компілятора, який отримав назву *AVS*, який перетворює програми, що не знають *DVS*, у програми, що знають про *DVS* [44]. Компілятор профілює програму під час компіляції та коментує інформацію про залишкові найгірші випадки циклів виконання (*RWEC*), яка представляє решту найгірших циклів виконання серед усіх шляхів виконання, які починаються з кожної відповідної контрольної точки. Він автоматизує розробку програм, що працюють у режимі реального часу, на процесорі зі змінною напругою, повністю прозорим для розробників програмного забезпечення.

У попередньо обговорених підходах масштабування напруги/частоти повинно обчислюватися і виконуватися в кожному контрольно-пропускному пункті, що може спричинити неконтрольовані витрати під час виконання. Газале та ін. повідомляв про подібний підхід на основі компілятора, але вимагає співпраці між компілятором та операційною системою [18]. Як і раніше, компілятор коментує пункти пропуску тимчасовою інформацією *RWEC*. Під час виконання програми операційна система періодично адаптує напругу та частоту процесора на основі цієї часової інформації. Тому такий підхід розділяє контрольні точки на дві категорії: перша використовується лише для обчислення часової інформації та коригування динамічної інформації про час виконання. Другий використовується ОС (яка має більше інформації про загальну поведінку програми) для виконання зміни напруги/частоти. Цей підхід спирається на сильні сторони як компілятора, так і ОС для отримання дрібної інформації про виконання програми для оптимального застосування *DVS*.

COPPER [1] - ще один підхід на основі компілятора, який також покладається на характеристики мікроархітектури для оптимізації енергоефективності програми. Серед

багатьох можливостей він зосереджений на поєднанні реконфігурації файлу динамічного реєстру з масштабуванням напруги/частоти. Під час компіляції різні версії даного програмного коду виробляються з різними архітектурними параметрами, в основному за кількістю доступних регістрів, а відповідні профілі потужності оцінюються за допомогою симулятора енергії, такого як *Wattch*. Оскільки запуск кодової версії, скомпільованої для меншої кількості регістрів, може призвести до нижчого споживання енергії, але більш високого виконання затримки, можна встановити компроміс між середнім споживанням енергії та часом виконання з версією коду. Система часу виконання вибирає версію коду, щоб допомогти досягти цілей продуктивності в рамках заданих енергетичних обмежень.

2.2. Повна система системного рівня *DPM*

Як вже обговорювалося раніше, центральний процесор не домінує в енергоспоживанні всієї системи. Інші системні компоненти, такі як дисководи та дисплеї, мають набагато більший внесок. Тому необхідно розглянути всі найважливіші компоненти системи, щоб ефективно оптимізувати потужність. Добре відома техніка управління живленням на системному рівні - це вимкнення жорстких дисків та дисплеїв, коли вони не працюють. Подібна ідея може бути застосована і до інших пристроїв вводу-виводу для економії енергії. Однак зміна станів живлення апаратних компонентів спричиняє не лише затримку часу, але й накладні витрати енергії. Отже, пристрій слід присипляти лише в тому випадку, якщо заощаджена енергія виправдовує накладні витрати. Таким чином, головна проблема успішного застосування цієї техніки полягає в тому, щоб знати, коли слід вимкнути пристрої та розбудити їх.

Простий метод полягає в тому, щоб окремі пристрої приймали такі рішення, контролюючи своє власне використання. Однією з явних переваг цієї схеми, що базується на пристроях, є прозорість, тобто економія енергії досягається без залучення або зміни прикладного або системного програмного забезпечення. Навпаки, ця схема

може працювати погано, оскільки вона не знає про завдання, що вимагають обслуговування пристрою. Для полегшення цієї проблеми були запропоновані програмні методи *DPM*. Прикладне або системне програмне забезпечення несе повну відповідальність за рішення, пов'язані з енергоспоживанням, припускаючи, що пристрої можуть працювати в декількох режимах низької потужності, використовуючи інтерфейси управління, такі як розширена конфігурація та інтерфейс живлення (*ACPI*) [74].

2.2.1. Політика *DPM* на основі апаратних пристроїв

Політика апаратного пристрою базується на апаратній діяльності та навантаженнях цільового пристрою та відповідно змінює стан живлення. Зазвичай вони реалізуються в апаратному забезпеченні або драйверах пристроїв без безпосередньої взаємодії з прикладним або системним програмним забезпеченням, як показано на рис.

2.4. На основі механізмів прогнозування майбутнього використання пристрою ці методи можна класифікувати на три категорії: час очікування, прогнозування та стохастична політика [7, 37].

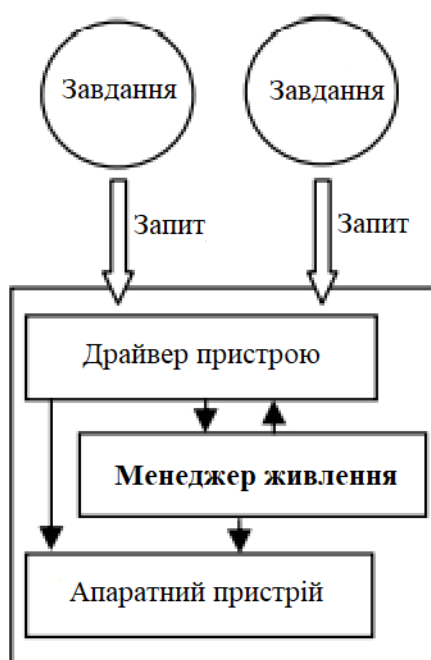


Рис. 2.4. *DPM* на основі апаратних пристроїв.

Що стосується політики тайм-ауту, час беззбитковості, T_{BE} , визначається як мінімальна тривалість часу періоду простою, протягом якого вимкнення певного пристрою буде економити енергію. При переході потужності, P_{TR} і час переходу, T_{TR} , (необхідні при зміні станів живлення пристрою) є поважний, T_{BE} дорівнює нулю, оскільки немає затримки та витрат енергії на вимкнення та пробудження на практиці, T_{BE} обчислюється на основі P_{TR} і T_{TR} як на рис 2.5. .

Працює наступним чином: коли починається період простою, запускається таймер із заданою тривалістю $T_{timeout}$, яка зазвичай встановлюється як певна частка T_{BE} . Якщо пристрій залишається без роботи після $T_{час}$ вийшов, тоді менеджер живлення робить перехід у режим низького енергоспоживання або вимкнений, оскільки передбачає, що пристрій залишатиметься в режимі очікування ще принаймні T_{BE} секунд. Цю політику порівняно просто зробити але вони мають два основних недоліки. По-перше, витрачається велика кількість енергії, чекаючи закінчення часу очікування, протягом якого пристрій перебуває в режимі очікування, але все ще повністю працює. По-друге, є завжди покарання за продуктивність пристроїв пробудження після отримання сигналу запиту. Типово пристрої мають тривалий час пробудження, і, отже, запит на пробудження може спричинити значні затримки.

T_{TR} : transition time required to enter ($T_{On,Off}$) and exit ($T_{Off,On}$) the inactive state

P_{TR} : transition power

P_{On}, P_{Off} : power when device is On and Off

T_{BE} : break-even time, the minimum length of an idle period during which shutting down the device will save power.

$$T_{TR} = T_{On,Off} + T_{Off,On}$$

$$P_{TR} = (T_{On,Off} P_{On,Off} + T_{Off,On} P_{Off,On}) / T_{TR}$$

$$T_{BE} = T_{TR} \quad \text{if } P_{TR} \leq P_{On}$$

$$T_{TR} + T_{TR} (P_{TR} - P_{On}) / (P_{On} - P_{Off}) \quad \text{if } P_{TR} > P_{On}$$

where “ $T_{TR} (P_{TR} - P_{On}) / (P_{On} - P_{Off})$ ” represents the additional time needed to spend in the Off state to compensate the excess power consumed during state transition.

Рис. 2.5. Розрахунок часу беззбитковості, T_{BE} [4].

Прогнозуюча політика протидіє недолікам політики тайм-ауту, використовуючи такі методи, як прогнозоване вимкнення [15, 17, 24, 60] та прогнозоване пробудження [24]. Політика інтелектуального вимкнення усуває період очікування, попередньо передбачивши тривалість простою. Шрівастава та інші припустили, що тривалість простою може бути передбачена тривалістю попереднього періоду зайнятості [47]. Чунг та ін. спостерігали закономірність періодів простою, і тоді тривалість поточного періоду простою прогнозується шляхом узгодження поточної послідовності, яка призвела до цього періоду простою, із спостережуваною історією періодів простою [12]. У роботах [17, 24] дослідники запропонували динамічно регулювати $T_{timeout}$ залежно від того, чи були попередні прогнози правильними чи ні. Прогнозована політика пробудження зменшує покарання за продуктивність, пробуджуючи пристрій вчасно, щоб він був готовий до отримання наступного запиту. Хван та ін. застосував підхід до експоненціального середнього (середньозваженого з експоненціальними значеннями ваги) для прогнозування часу пробудження на основі минулої історії [24]. Ця політика може збільшити споживання енергії, але зменшить затримку обслуговування першого запиту після періоду простою.

Одним з недоліків політики прогнозування є те, що вони передбачають детермінований прихід запитів пристроїв. Стохастичні політики моделюють надходження запитів та зміни стану пристрою як стохастичні процеси, наприклад, процеси Маркова. Беніні та ін. змоделивав надходження запитів вводу-виводу за допомогою стаціонарних процесів Маркова з дискретним часом [5]. Ця модель була використана для досягнення оптимальної економії енергії шляхом вимкнення та пробудження пристрою найбільш ефективним способом як з точки зору енергії, так і продуктивності. У цій моделі час ділиться на невеликі інтервали з припущенням, що система може змінити свій стан лише на початку інтервалу часу. Чунг та ін. розширив модель за рахунок розгляду нестаціонарних процесів [11]. Вони попередньо обчислили оптимальний графік для різних шаблонів запитів вводу/виводу, і під час виконання цих

графіків вони використовуються для більш точної оцінки часу наступного запиту вводу/виводу.

Однак для моделей Маркова з дискретним часом менеджера живлення необхідно надсилати контрольні сигнали до компонентів через кожний інтервал часу, що може призвести до інтенсивного руху сигналу і, отже, до більшого розсіювання потужності. Кюи та співавт. використовував моделі Маркова безперервного часу, щоб запобігти цьому “періодичному оцінюванню”, і замість цього використовує оцінку, спричинену подією [41]. Вони розглядають як запит прибуття, так і запит на події обслуговування, на яких система визначає, чи слід вимикати пристрій. Нарешті, Сімунік та інші запропонував додати час очікування до моделей Маркова безперервного часу, щоб пристрій було вимкнено, якщо воно безперервно перебувало в режимі очікування протягом заздалегідь визначеної тривалості часу очікування [45]. Загалом, стохастичні політики забезпечують кращу ефективність, ніж політики прогнозування та тайм-ауту. Крім того, вони здатні управляти різними станами потужності, приймати рішення не тільки про те, коли здійснювати перехід станів, але й про те, який перехід слід зробити. Основним недоліком цих політик є те, що вони вимагають попередньої обробки в автономному режимі і є більш складними для реалізації. Детальне порівняння вищезазначених схем *DPM* на основі пристроїв див. у [7, 37].

2.2.2 Програмні політики *DPM*

Хоча політики керування живленням на основі апаратних пристроїв можуть оптимізувати енергоефективність окремих пристроїв, вони не враховують загальносистемне споживання енергії через відсутність глобальної інформації. Тому пропонуються програмні політики *DPM* для управління системним управлінням живленням. Розширений інтерфейс конфігурації та живлення (*ACPI*) [74], запропонований *Intel*, *Microsoft* та *Toshiba* як промисловий стандарт, забезпечує програмно-апаратний інтерфейс, що дозволяє менеджерам живлення контролювати

потужність різних компонентів системи. Методи *DPM* на основі застосування та операційної системи, які будуть розглянуті далі в цьому підрозділі, використовують такий інтерфейс для економії енергії. Хоча схеми на основі додатків можуть бути найефективнішими, оскільки майбутні робочі навантаження є найбільш відомими додаткам, схеми на базі ОС мають перевагу, полягає в тому, що для економії енергії не потрібно переписувати існуючі програми.

2.2.2.1. Розширений інтерфейс налаштування та живлення (*ACPI*)

ACPI [74] еволюціонував із старішого стандарту *APM* для націлювання на настільні ПК [75]. Політики *APM*, реалізовані на рівні *BIOS*, досить прості та детерміновані. Програма та ОС здійснюють звичайні дзвінки *BIOS* для доступу до пристрою, а *APOS*-обізнаний *BIOS* обслуговує запити вводу-виводу, заощаджуючи енергію. Однією з переваг *APM* є те, що весь процес є прозорим для прикладного програмного забезпечення та програмного забезпечення ОС. *ACPI* є заміною *APM* на рівні системного програмного забезпечення. На відміну від *APM*, *ACPI* безпосередньо не займається управлінням живленням. Замість цього він надає ОС інтерфейси управління живленням для різних апаратних пристроїв, а відповідальність за управління живленням покладається на програмне забезпечення операційної системи. На рис. 2.6 наведено огляд взаємодії між компонентами системи в *ACPI* [74]. Фронтенд *ACPI* - це драйвер пристрою, сумісний з *ACPI*. Він відображає запити ядра на команди *ACPI* та відображає відповіді *ACPI* на переривання вводу-виводу або сигнали ядра. Зауважте, що ядро може також взаємодіяти з несумісним з *ACPI* обладнанням через інші драйвери пристроїв.

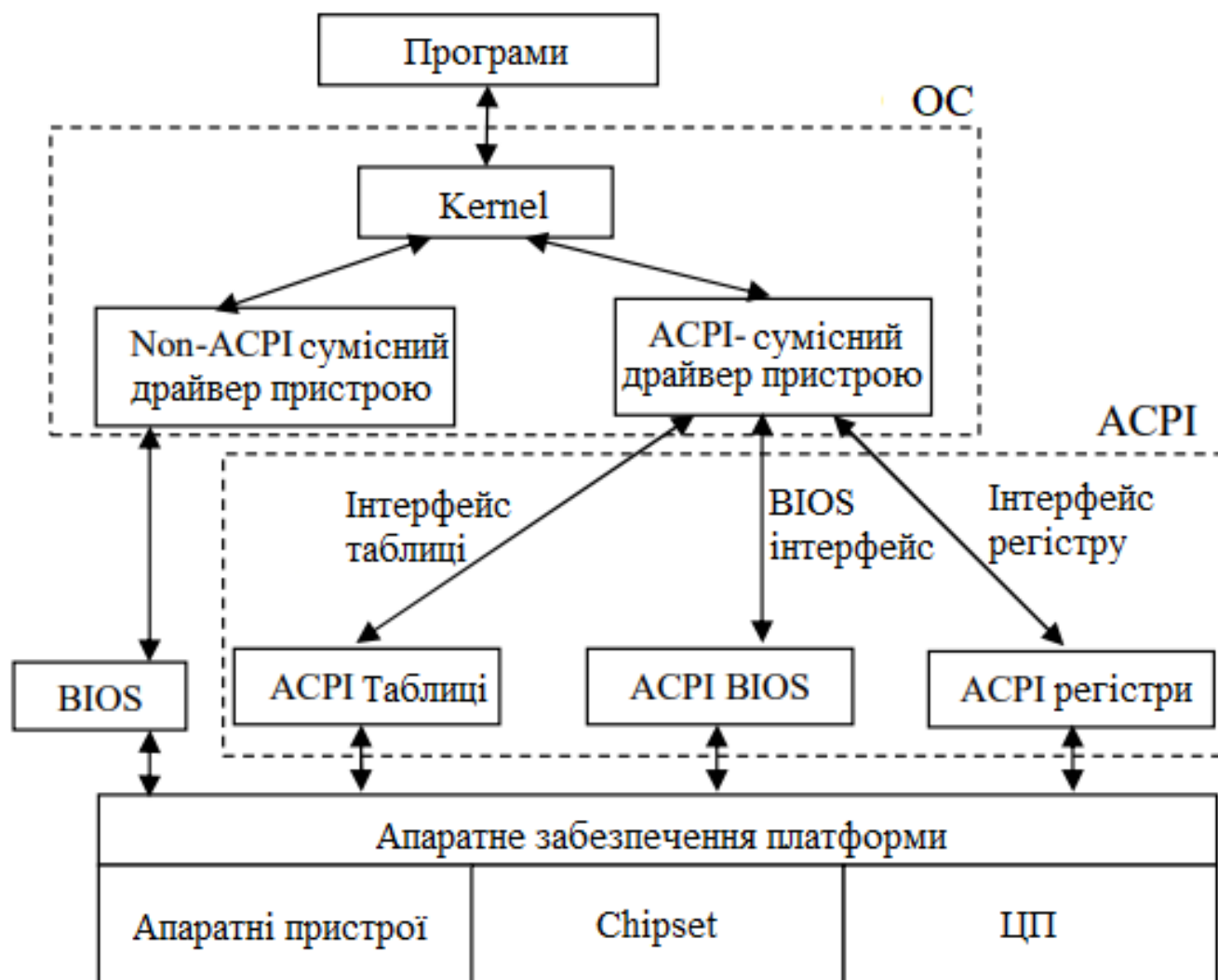


Рис. 2.6. Взаємодія між компонентами системи з *ACPI*

Специфікація *ACPI* визначає п'ять глобальних системних станів: *G0* представляє робочий стан, *G1 - G3* - сплячі стани і, нарешті, один застарілий стан для пристроїв, що не відповідають *ACPI*. Специфікація також уточнює стан сну, визначаючи додаткові чотири стану сну (*S1-S4*) у стані *G1*, як показано в табл. 2.5. На додаток до глобальних станів, *ACPI* також визначає чотири стану пристрою (*D0-D3*) та чотири процесора (*C0-C3*). Різні стани відрізняються між собою енергією, яку вони споживають, і часом, необхідним для пробудження. Наприклад, стан глибокого сну, такий як стан *S4* у табл. 2.5, економить більше енергії, але пробудження займає більше часу.

Глобальні стани *ACPI*

Стани		Опис
G3		Механічне вимкнення: відсутність споживання енергії, вимкнення системи
G2		Програмне вимкнення: повне перезавантаження ОС потрібно для відновлення робочого стану.
G1	Стани сну	Сплячий: система, здається вимкнена і повернеться до робочого стану через час, що збільшується із зворотнім споживанням енергії.
	S4	Найдовша затримка пробудження та найменша потужність. Усі пристрої вимкнені.
	S3	Низька затримка пробудження. Усі системні контексти втрачаються, крім системної пам'яті.
	S2	Низька затримка пробудження. Втрачається лише контекст процесора та системного кешу.
	S1	Найнижча затримка пробудження. Жоден системний контекст не втрачається.
G0		Працює: система увімкнена і повністю працює.
Політика		Політика: вводиться, коли система не відповідає <i>ACPI</i> .

2.2.2.2. *DPM* на основі додатків

Політика *DPM* на основі додатків стала можливою завдяки появі вищезгаданого стандартного стану *ACPI*. Ці політики переміщують диспетчер живлення з рівня пристрою чи обладнання на рівень програми. Програма, яка є джерелом більшості запитів до цільового пристрою, тепер відповідає за управління станами живлення цього пристрою. Ці політики дозволяють прикладним програмам переводити пристрій у

повністю робочий стан, надсилати його в сплячий режим, пробуджувати або отримувати повідомлення про зміни стану живлення пристрою. Наприклад, *Microsoft OnNow* [81] та *ACPI4Linux* [73] підтримують управління живленням для пристроїв, сумісних з *ACPI*.

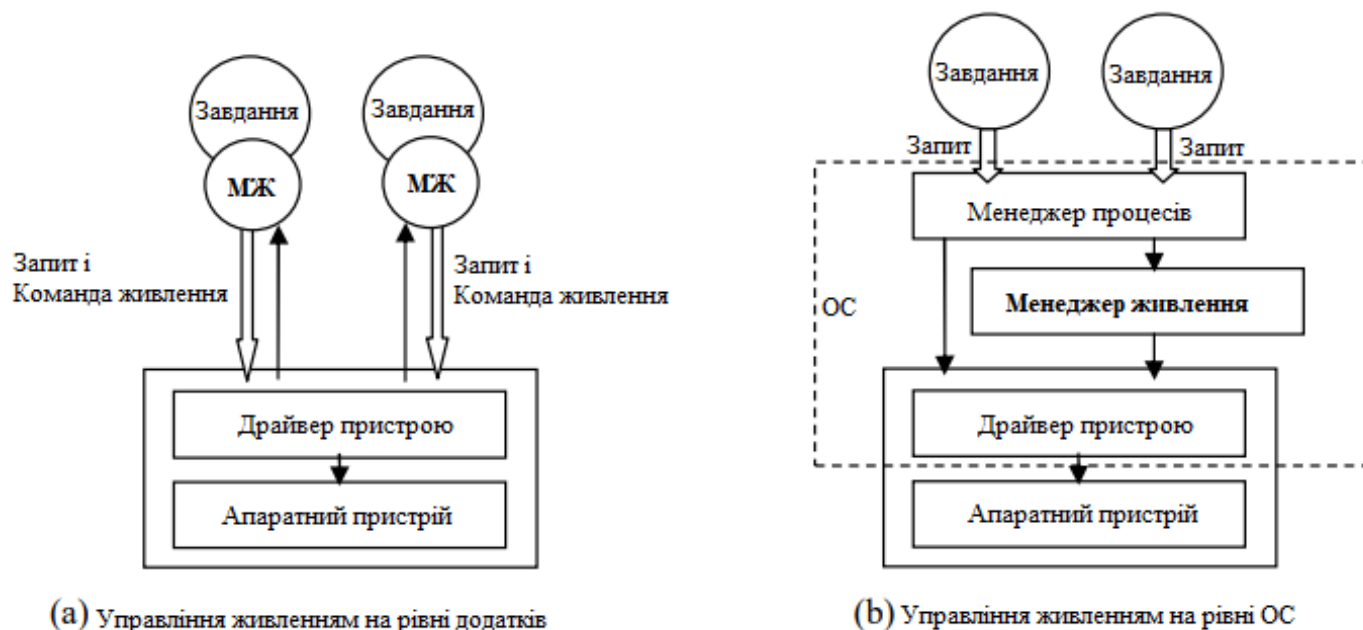


Рис. 2.7. Політики *DPM* на основі програмного забезпечення

Альтернативно, Лу та інші запропонували програмну архітектуру, яка експортує механізми управління живленням на рівень додатків за допомогою шаблону [32]. Ця схема приймає рішення про стан живлення на основі інформації про параметри системи, такі як потужність у кожному стані, енергія переходу, затримка та майбутнє навантаження. Ця архітектура програмного забезпечення відрізняється від методів, заснованих на *ACPI*, тим, що управління живленням централізоване до однієї програми, що робить її безпечнішою та ефективнішою в одній системі додатків.

2.2.2.3. *DPM* на основі операційної системи

Управління живленням на основі додатків має кілька недоліків. По-перше, вони вимагають модифікацій існуючих додатків, і, отже, реалізація цих політик покладе додаткове навантаження на програмістів. По-друге, технологічний прогрес постійно

змінює апаратні параметри, що робить політику, оптимізовану для певного середовища, неефективною після заміни пристрою. Нарешті, різні програми можуть встановлювати один і той же пристрій на різні стани живлення, що спричиняє нестабільність системи. Методи *DPM* на основі ОС використовують знання операційної системи про всі запуснені процеси та їх взаємодію з апаратним забезпеченням для оптимізації енергоефективності. У літературі запропоновано ряд схем *DPM* на основі ОС [23, 35, 36]. Рис. 2.7 (b) ілюструє реалізацію управління живленням на основі ОС.

Лу та ін. запропонував керування живленням на основі завдань, яке використовує ідентифікатори процесів на рівні ОС для диференціації завдань, які роблять запити вводу-виводу [35, 36]. Це має головну перевагу перед політиками, заснованими на пристроях, тим, що пропонує краще розуміння використання пристрою, а також його майбутнього способу використання. Наприклад, схема *DPM* на основі ОС, яка називається енергоорієнтованим плануванням процесів, планує завдання, кластеризуючи періоди простою, щоб зменшити кількість переходів стану та накладних витрат на перехід стану [29]. Нарешті, Гнайді та інші запропонував використовувати програмні лічильники для прогнозування операцій вводу/виводу в операційній системі [19]. Їх передбачувач доступу до лічильника програм динамічно вивчає шаблони доступу програми, використовуючи кореляцію на основі шляху, щоб відповідати певній кількості лічильників програм, що ведуть до кожного періоду простою. Потім ця інформація використовується для прогнозування майбутніх випадків цього простою і, таким чином, оптимізації потужності.

2.3. Паралельний *DPM* на системному рівні

Більшість тем, що стосуються енергетики, присвячені уніпроцесорним системам. Однак, завдяки кооперативному характеру обчислень у паралельних обчислювальних середовищах, найбільш енергоефективне виконання для кожного окремого процесора не обов'язково може призвести до найкращого загального енергоефективного виконання.

Зменшення енергоспоживання не тільки зменшує експлуатаційні витрати на охолодження, але й підвищує надійність, що часто є критично важливим для цих висококласних систем. У цьому розділі представлені методи *DPM* для паралельних систем, запропоновані в літературі. По-перше, апаратні методи оптимізації живлення, такі як скоординований *DVS* [15] та мережі міжпотужного з'єднання малої потужності [32, 53] в кластерних системах представлені в підрозділі 2.3.1. По-друге, програмні технології *DPM*, такі як енергозберігаюча синхронізація для багатопроцесорних систем [28], представлені в підрозділі 2.3.2. У цьому підрозділі також представлені техніки *DPM*, що використовуються в кластерах серверів для зменшення енергоспоживання всього кластера шляхом координації та розподілу робочого навантаження між усіма доступними вузлами [40].

2.3.1. Апаратні технології *DPM*

2.3.1.1. Узгоджене динамічне масштабування напруги (*CVS*)

Елнзоахи та ін. також запропонував використовувати схему *DVS*, обговорену в підрозділі 2.1.3, у кластерній системі [15]. Вони представили п'ять таких політик для порівняння. Перша політика, незалежне масштабування напруги (*IVS*), просто використовує процесори з масштабуванням напруги, де кожен вузол самостійно керує власним споживанням енергії. Друга політика, яка називається координованим масштабуванням напруги (*CVS*), використовує *DVS* скоординовано, так що всі вузли кластера працюють дуже близько до налаштування середньої частоти по кластеру, щоб зменшити загальну вартість енергії. Цього можна досягти шляхом періодичного обчислення налаштування середньої частоти всіх активних вузлів центральним монітором і передає його на всі вузли кластера. Третя політика, яка називається *VOVO*, вимикає деякі вузли, так що залишається в живих лише мінімальна кількість вузлів, необхідних для підтримки робочого навантаження. Четверта політика, яка називається

комбінованою політикою, поєднує в собі IVS та VOVO, тоді як п'ята політика, яка називається координованою політикою, використовує комбінацію CVS та VOVO. За їхньою оцінкою, дві останні політики пропонують найбільшу економію енергії. Серед них узгоджена політика (CVS-VOVO) економить більше енергії за рахунок більш складного впровадження.

2.3.1.2. *DPM* на основі мережевих взаємозв'язків

Одним з найбільш критичних витрат енергії в паралельних системах є комунікаційні зв'язки між вузлами, що є важливим фактором розмежування порівняно з однопроцесорними системами. Засоби зв'язку (комутатори, шини, мережеві карти тощо) споживають велику кількість енергетичного бюджету кластерної системи [32, 53], що особливо актуально при зростаючому попиті на пропускну здатність мережі в таких системах. Шан та ін. запропонував застосовувати *DVS* до мережних мереж [43]. Інтуїція полягає в тому, що якщо можна точно налаштувати пропускну здатність мережі, щоб слідувати використанню посилання, можна досягти величезної економії енергії. Політика *DVS*, що базується на історії, використовує минулий мережевий трафік з точки зору використання лінії зв'язку та використання вхідного буфера вхідного сигналу для прогнозування майбутнього трафіку.

Кім та співавт. Запропонували альтернативну схему *DPM*, застосовану до ліній взаємозв'язку. Вони розглянули потенційну проблему погіршення роботи, особливо у ситуаціях із низьким та середнім навантаженням. Це може призвести до більшого використання буфера, що також збільшує загальне споживання енергії витоків. Їх метод, який називається схема динамічного вимкнення посилань (*DLS*), намагається полегшити проблему, виходячи з того, що підмножина недостатньо використовуваних посилань (з використанням під певним порогом) може бути повністю закрита, припускаючи, що інший піднабір високо використовуваних посилань може для забезпечення зв'язку в мережі.

2.3.2. Програмні технології *DPM*

2.3.2.1. *DPM* на основі бар'єрних операцій

Як обговорювалося в попередньому підрозділі, лінії взаємозв'язку можуть бути найбільш критичним вузьким місцем у паралельних системах щодо споживання енергії, а також обчислювальних характеристик. З точки зору прикладного програмного забезпечення, колективні комунікації, такі як бар'єри, часто вважаються найбільш критичним вузьким місцем під час виконання паралельного додатка [35]. У звичайній багатопроесорній системі ранній потік зупиняється (як правило, шляхом очікування обертання) біля бар'єру і чекає, поки всі повільніші потоки прийдуть, перш ніж продовжувати виконання за бар'єром. Цей бар'єрний спін-очікування є дуже неефективним, оскільки витрачається потужність при виконанні непродуктивних обчислень.

Лі та ін. запропонував ощадливий бар'єр [28], коли нитка, що надходить, намагається перевести свій процесор у стан низької потужності, а не просто обертатися. Коли приходить остання нитка, сплячі процесори прокидаються, і всі нитки проходять повз бар'єр. Однак, як обговорювалося раніше в підрозділі 2.2.1, переходи стану енергії повинні виправдовувати економію енергії в порівнянні з часом затримки, що виникає в процесі. Отже, кожен потік, який приходить раніше, повинен передбачити тривалість очікуваного часу зупинки та вирішити, чи переходити до стану низької потужності чи ні, і якщо так, то вибрати найкращий стан низької потужності. У той же час, час пробудження також слід передбачити, щоб компенсувати економію енергії проти погіршення продуктивності. Щоб вирішити ці проблеми, ощадливий бар'єр використовує минулу історію інтервального часу між двома послідовними бар'єрами для прогнозування часу зупинки на бар'єрі [28]. Основна мета полягає в тому, щоб пробудити сплячі нитки саме вчасно для подальшого виконання, тим самим досягнувши значної економії енергії, не спричиняючи погіршення продуктивності.

2.3.2.2. *DPM* з балансуванням навантаження

У кластерній системі балансування навантаження - це техніка, яка використовується для рівномірного розподілу робочого навантаження по всіх доступних вузлах таким чином, щоб ефективно використовувалися всі простой. Пінейро та ін. використовували концепцію розбалансування навантаження для зменшення енергоспоживання кластерної системи [40]. На відміну від балансування навантаження, він концентрує роботу в меншій кількості вузлів, а на холостому ходу інших, які можна вимкнути, що призведе до економії енергії, але в той же час може погіршити продуктивність. Їх алгоритм періодично оцінює, чи слід видаляти або додавати деякі вузли до кластера на основі прогнозованого енергоспоживання та заданого загального навантаження, накладеного на кластер з різними конфігураціями кластера. Якщо вузли використовуються недостатньо, деякі з них будуть видалені, а якщо вузли будуть надмірно використані, слід додати нові вузли. В обох випадках алгоритм перерозподіляє наявне навантаження між активними вузлами кластера. Повідомлялося про значне зниження потужності лише із незначним погіршенням продуктивності [40].

Висновки до розділу

Методи *DPM* націлені на зменшення споживання енергії під час роботи шляхом вибіркового вимкнення або уповільнення компонентів, коли системи простоюють або обслуговують невеликі робочі навантаження. Методи *DPM* застосовуються по-різному та на різних рівнях. Вони відстежують навантаження системи, щоб передбачити майбутні вимоги до обчислень і намагаються динамічно адаптувати поведінку системи відповідно.

Були розглянуті методи *DPM*, класифіковані на основі рівня впровадження:

- на рівні ЦП;

- на системному рівні, що розглядають інші системні компоненти (пам'ять, жорсткий диск, пристрої вводу-виводу, дисплей тощо), ніж ЦП;
- методи *DPM*, запропоновані для паралельних систем, де кілька вузлів співпрацюють, щоб заощадити загальну потужність, колективно виконуючи задане паралельне завдання.

РОЗДІЛ 3

ЗНИЖЕННЯ ЕНЕРГІЇ У СПІЛКУВАННІ

До цього моменту ми в основному обговорювали методи, які можна використовувати для зменшення енергоспоживання цифрових систем, і зосетрилися на комп'ютерних системах. У цьому підрозділі ми обговоримо деякі методи, які можуть бути використані для зменшення споживання енергії, необхідної для (бездротового) зв'язку поза комп'ютером.

3.1. Джерела споживання енергії

У найбільш абстрактному вигляді мережева комп'ютерна система має два джерела витрати енергії під час роботи.

Спілкування, за рахунок енергії, витраченої бездротовим інтерфейсом. Енергія зв'язку, серед іншого, диктується вимогами відношення сигнал/шум (SNR).

Обчислення, внаслідок обробки (сигналу) та інших завдань, необхідних під час спілкування. Обчислювальна енергія - це функція апаратного та програмного забезпечення, що використовується для таких завдань, як стиснення та виправлення помилок вперед (FEC).

Взагалі кажучи, мінімізація споживання енергії є завданням, яке вимагатиме мінімізації внесків зв'язку та обчислень, внесення відповідних компромісів між ними. Наприклад, зменшення обсягу переданих даних може бути корисним. З іншого боку, вартість обчислення (наприклад, для стиснення даних, що надсилаються) може бути високою, і в крайньому випадку вона може бути такою, що було б краще просто надіслати необроблені дані.

У міжміських бездротових лініях домінує енергетична складова передавального зв'язку. Однак для бездротових ліній зв'язку на короткі відстані та в суворих умовах, де

може використовуватися велика кількість обробки сигналів і обчислення протоколів, обчислювальний компонент може бути значним або домінуючим.

Інтерфейс бездротової мережі мобільного комп'ютера споживає значну частку загальної потужності [58]. Вимірювання показують, що в типових додатках, таких як перегляд веб-сторінок або обробка електронної пошти, енергія, яка споживається в той час, коли інтерфейс увімкнений і не працює, перевищує вартість фактичного отримання пакетів. Це тому, що більшість програм мають мало вимогливі потреби в трафіку, а отже, трансивер є холостий хід більшість часу. Доступ до бездротового каналу контролюється за протоколом *MAC*. Багато протоколів *MAC* для бездротових мереж є в основному адаптацією протоколів *MAC*, що використовуються в дротових мережах, і ігнорують енергетичні проблеми [59]. Наприклад, протоколи *MAC* з довільним доступом, такі як розпізнавання багаторазового доступу несучої з уникненням зіткнень (*CSMA/CA*) та 802.11, як правило, вимагають постійного ввімкнення приймача та моніторингу каналу за трафіком. Типовий поріг бездіяльності, тобто час, перш ніж трансивер перейде у вимкнений або очікуваний стан після періоду бездіяльності, спричиняє необхідність перебування приймача в енергоємному режимі протягом значного часу. Значний час та енергія додатково витрачаються мобільним пристроєм на перемикання з режимів передачі на прийом та навпаки. В ефірних мережах зіткнення може виникнути (під час великих навантажень). Це призводить до того, що дані стають марними, а енергія, необхідна для транспортування цих даних, марнується.

Наступним кроком є зменшити обсяг даних, який потрібно проштовхнути через канал. Цієї мети можна досягти різними шляхами. Один з них - зменшити накладні витрати на протокол що впливає на енергетичні потреби через кількість «марних» даних управління та необхідних обчислень для обробки протоколів. Високий коефіцієнт помилок характерне для бездротових ліній зв'язку - це ще одне джерело споживання енергії з кількох причин. По-перше, коли дані неправильно отримують енергію, необхідну для транспортування та обробки цих даних, псується. По-друге, енергія використовується для механізмів контролю помилок. Нарешті, оскільки в бездротовому

зв'язку частота помилок динамічно змінюється в часі та просторі, механізм управління помилками з фіксованою точкою, розроблений таким чином, щоб мати можливість виправляти помилки, які майже не трапляються, псує енергію та пропускну здатність. Якщо додаток стійкий до помилок, спроба протистояти всім можливим помилкам псує ще більше енергії для непотрібного контролю помилок. Зменшення обсягу даних також є проблемою на рівні програми. Наприклад, програма може змінити швидкість стиснення або, можливо, зменшити роздільну здатність даних. Замість того, щоб надсилати ціле велике повнокольорове зображення, можна надсилати чорно-білі напіврозмірні зображення із стиском із втратами.

3.2. Стек мережевого протоколу

Протоколи передачі даних регулюють спосіб обміну інформацією електронними системами, вказуючи набір правил, які при дотриманні забезпечують послідовну, повторювану та добре зрозумілу послугу передачі даних. При розробці протоколів зв'язку та систем, що їх реалізують, хотілося б переконатися, що протокол правильний та ефективний.

Портативні пристрої мають суворі обмеження щодо розміру, енергоспоживання, доступної пропускну здатності зв'язку та необхідні для обробки багатьох класів передачі даних через бездротове з'єднання з обмеженою пропускну здатністю, включаючи трафік у режимі реального часу, сприйнятливий до затримок, наприклад, швидкість та відео. Мультимедійні програми характеризуються різними медіапотокami. Кожен потік може мати різні вимоги до якості обслуговування. Залежно від класу обслуговування та якості обслуговування з'єднання, програма може застосовувати до протоколу зв'язку іншу політику, щоб мінімізувати споживання енергії. Наприклад, уникаючи накладних витрат на управління помилками для з'єднань, які цього не потребують, і ніколи не передаючи застарілих даних, ефективність покращується. Ця комбінація обмеженої пропускну здатності, високого рівня помилок, а дані, що

враховують затримки, вимагають щільної інтеграції всіх підсистем у пристрій, включаючи агресивну оптимізацію протоколів, які відповідають запланованому застосуванню. Протоколи повинні бути надійними на наявність помилок; вони повинні вміти розрізняти класи даних, надаючи кожному класу точну послугу, яка йому потрібна; і вони повинні мати реалізацію, придатну для малопотужних портативних електронних пристроїв.

З метою економії енергії нормальним режимом роботи мобільного пристрою буде режим сну або відключення живлення. Для підтримки повного зв'язку, перебуваючи в режимі глибокого відключення, мережеві протоколи потрібно змінити. Схеми збереження та пересилання для бездротових мереж, такі як запропонований *IEEE 802.11* режим сну, не тільки дозволяють мережевому інтерфейсу переходити в режим сну, але також можуть виконувати локальні ретрансляції, не залучаючи вищі рівні мережевого протоколу. Однак такі схеми мають той недолік, що вимагають від третьої сторони, наприклад базової станції, виконувати функції буферного інтерфейсу. Однак цей приклад показує, що мережеві протоколи бездротової системи можуть бути змінені таким чином, що це мінімізує споживання енергії.

На міркування щодо енергоефективності принципово впливає компроміс між споживанням енергії та досяжною якістю обслуговування (*QoS*). Забезпечивши універсальний роумінг, користувач мобільного зв'язку зіткнеться із середовищем, в якому якість послуг може істотно відрізнятися в різних бездротових мережах та між ними. Для того, щоб витончено боротися з динамічними варіаціями мережевих та обчислювальних ресурсів, як мобільне обчислювальне середовище, так і програми, що працюють в такому середовищі, повинні адаптуватися їх поведінка залежно від наявних ресурсів, включаючи акумулятори. Зниження енергії слід враховувати у всій системі мобільного зв'язку та на всіх рівнях стеку протоколів, включаючи рівень додатків. Адаптивність протоколів є ключовим питанням. Зараз ми запропонуємо різні способи, які можна використовувати для зменшення споживання енергії на рівнях типового стеку мережевих протоколів.

Фізичний рівень - на найнижчому рівні нам потрібно застосувати енергоефективне радіо, яке може перебувати в різних режимах роботи (наприклад, змінна потужність RF та різні режими сну), щоб це дозволяло динамічно керувати енергією. Енергію також можна заощадити, якщо вона здатна адаптувати свої методи модуляції та основні схеми виправлення помилок. Пропускна здатність, яку пропонує радіо, також впливає на споживання енергії. Енергія на переданий або прийнятий біт, як правило, стає нижчою при більш високих бітових швидкостях. Наприклад, радіо *WaveLAN* працює зі швидкістю 2 Мбіт/с і споживає 1,8 Вт або 0,9 μ Дж/біт. Комерційно доступний FM-приймач (*Radiometrix BIM-433*) працює зі швидкістю 40 кбіт/с і споживає 60 мВт, або 1,5 μ Дж/біт. Це робить радіостанцію з низькою швидкістю передачі даних менш ефективною у споживанні енергії для того ж обсягу даних. Однак, коли мобільний телефон повинен довше слухати трансляцію або пробудження від базової станції, тоді радіо з високою швидкістю передачі споживає приблизно в 30 разів більше енергії, ніж радіо з низькою швидкістю передачі даних. Тому радіостанцію з низькою швидкістю передачі даних слід використовувати лише для базової сигналізації та якомога менше для передачі даних.

Щоб мінімізувати споживання енергії, а також зменшити перешкоди та збільшити пропускну здатність мережі, потужність передачі на лінії зв'язку повинна бути мінімізована, якщо це можливо.

Середній рівень доступу - в енергоефективному протоколі *MAC* основною метою є мінімізація всіх дій мережевого інтерфейсу, тобто мінімізація “часу роботи” передавача, а також приймача. Інший спосіб зменшити споживання енергії - мінімізувати кількість переходів, які повинен зробити бездротовий інтерфейс. Шляхом планування передачі даних масово, неактивному терміналу дозволяється дрімати і вимикати приймач, доки мережевий інтерфейс буде повторно активований у запланований час для передачі даних на повній швидкості. Прикладом енергоефективного протоколу *MAC* є E^2 MaC [60]. Це протокол TDMA, в якому

менеджер *QoS* на базовій станції планує весь трафік відповідно до вимог *QoS* і намагається мінімізувати енергоспоживання мобільних телефонів.

Рівень управління логічним посиленням - через динамічний характер бездротових мереж, адаптивний контроль помилок дає значний вигаш у пропускній здатності та енергоефективності [61] [62]. Це дозволяє уникнути застосування накладних витрат на управління помилками до з'єднань, які цього не потребують, і це дозволяє вибірково відповідати необхідним *QoS* та умовам радіолінії. Над цими адаптаціями контролю помилок планувальник на базовій станції може також адаптувати планування трафіку до умов помилок бездротових з'єднань з мобільним. Планувальник може спробувати уникнути періодів поганих умов помилок, не плануючи критичний трафік, що не є часом, протягом цих періодів.

Механізми регулювання потоку потрібні для запобігання переповненню буфера, а також для відкидання пакетів, які перевищили допустимий час передачі. Мультимедійні програми характеризуються різними медіапотоками. Кожен потік може мати різні вимоги до якості обслуговування. Залежно від класу обслуговування та якості обслуговування з'єднання може застосовуватися інший контроль потоку, щоб він мінімізував необхідну пропускну здатність та споживання енергії. Наприклад, у відеопрограмі марно передавати зображення, які вже застаріли. Важливіше мати "свіжі" зображення. Для такого трафіку буфер, мабуть, невеликий, і коли з'єднання заважає десь найдавніші дані будуть відкинуті, а свіжі дані будуть переміщені у *fifo*. Контроль потоку витратив би енергію на передачу «старих» зображень та повідомлень про управління потоком. Енергоефективне регулювання потоку пристосовує його механізм управління до вимог з'єднання.

Мережевий рівень - помилки в бездротовому каналі можуть поширюватися в стеку протоколів. За наявності високої частоти помилок пакетів і періодів періодичного з'єднання бездротових ліній зв'язку деякі мережеві протоколи (наприклад, *TCP*) можуть надмірно реагувати на втрати пакетів, приймаючи їх за перевантаження. *TCP* реагує на всі втрати, використовуючи алгоритми контролю загрози та уникнення. Ці заходи

призводять до непотрібного зменшення використання смуги пропускання лінії зв'язку та збільшення споживання енергії, оскільки це призводить до більш тривалого часу передачі. Обмеження *TSP* можна подолати за допомогою більш адекватного контролю перевантаження під час помилок пакетів. Ці схеми вибирають з безлічі механізмів для поліпшення наскрізної пропускну здатності, таких як локальні ретрансляції, розділені з'єднання та пряме виправлення помилок. У [63] було розглянуто та порівняно кілька схем. Ці схеми класифікуються на три категорії: наскрізні протоколи, коли відправник знає про бездротове з'єднання; протоколи рівня зв'язку, які забезпечують локальну надійність і захищають відправника від бездротових втрат; і протоколи розділеного з'єднання, які розбивають наскрізне з'єднання на дві частини на базовій станції. Їх результати показують, що надійний протокол рівня зв'язку з певними знаннями *TSP* забезпечує хорошу продуктивність, ніж використання підходу з розділеним з'єднанням. Схеми вибіркового підтвердження корисні, особливо коли втрати трапляються сплесками, які розривають наскрізне з'єднання на дві частини на базовій станції. Їх результати показують, що надійний протокол рівня зв'язку з певними знаннями *TSP* забезпечує хорошу продуктивність, ніж використання підходу з розділеним з'єднанням. Схеми вибіркового підтвердження корисні, особливо коли втрати трапляються сплесками, які розривають наскрізне з'єднання на дві частини на базовій станції. Їх результати показують, що надійний протокол рівня зв'язку з певними знаннями *TSP* забезпечує хорошу продуктивність, ніж використання підходу з розділеним з'єднанням. Схеми вибіркового підтвердження корисні, особливо тоді, коли втрати відбуваються сплесками.

Рівень операційної системи - інший спосіб запобігти високій вартості (або продуктивності, і енергоспоживання, і грошей) бездротового мережевого зв'язку - це уникнути використання мережі, коли вона дорога, передбачаючи майбутній доступ та отримуючи необхідні дані, коли мережа дешева. У протоколах вищого рівня системи зв'язку кешування та планування можуть використовуватися для управління передачею повідомлень. Це особливо добре працює, коли комп'ютерна система має можливість

використовувати різні мережеві інфраструктури (залежно від наявності інфраструктури в певному населеному пункті), з різним і багаторазовим підключенням до мережі та з різними характеристиками та витратами [64]. Справжнє передбачення, звичайно, вимагає знання майбутнього. Два можливі прийоми, керування *LRU* і накопичення, наприклад, присутні в кеш-менеджері *Coda* [65]. Для ефективної підтримки мобільних комп'ютерів розробники систем повинні розглядати мережу як першокласний ресурс, витрачаючи центральний процесор і, можливо, дискові ресурси, щоб зменшити використання мережевих ресурсів у періоди поганого мережевого зв'язку.

Сучасні високопродуктивні мережеві протоколи вимагають, щоб весь доступ до мережі здійснювався через операційну систему, що додає значні накладні витрати як на шлях передачі (як правило, системний виклик і копіювання даних), так і на шлях прийому (зазвичай переривання, системний дзвінок і копія даних). Це не тільки спричиняє проблеми з роботою, але й значне споживання енергії. Розумні мережеві інтерфейси можуть деяким чином усунути цю проблему. Для вирішення проблеми продуктивності були розроблені кілька архітектур комунікації на рівні користувача, які вилучають операційну систему з критичного шляху зв'язку [66].

3.3. Протоколи MAC

WSN можна описати як мережу датчиків, які взаємодіють між собою бездротовим способом. Ці датчики можуть бути встановлені в автоматичному середовищі з обмеженими обчислювальними та сенсорними можливостями. Отже, вони повинні бути стійкими до несправностей та надійними, щоб вимоги до технічного обслуговування були меншими. Щоб продовжити термін служби мережі, витрати енергії повинні бути мінімальними. Енергозбереження може бути здійснено за допомогою ефективного макропрограмування *WSN*. Протоколи контролю доступу середнього рівня (*MAC*) відіграють важливу роль у енергозбереженні. У цьому підрозділі описуються протоколи *MAC* на основі *CSMA* для *WSN* та аналізуються результати моделювання цих протоколів.

Були впроваджені *S-MAC*, *T-MAC*, *B-MAC*, *B-MAC +*, *X-MAC*, *DMAC* та *Wise-MAC* в TOSSIM симуляторі, який на відміну від інших симуляторів імітує той самий код, що працює на реальному обладнанні.

3.3.1. *S-MAC*

Sensor-MAC (*SMAC*) [67] - це протокол на основі суперечок, який регулює періоди сну в сенсорній мережі для економії енергії та поліпшення терміну служби мережі. Цей протокол представляє базову лінію орієнтованих на сон енергоефективних конструкцій протоколів *WSN MAC*. З чотирьох методів уникнення простою: статичне планування сну, динамічне планування сну, вибірки преамбули та офлайн-планування, *SMAC* застосовує статичне планування сну для збереження енергії. *SMAC* ділить час на кадри. Кожен кадр ділиться на активний та період сну, як показано на рис. 3.1. В активний період передавач-приймач вмикається і вимикається під час сну. Активний період додатково ділиться на період синхронізації часу та період передачі даних.

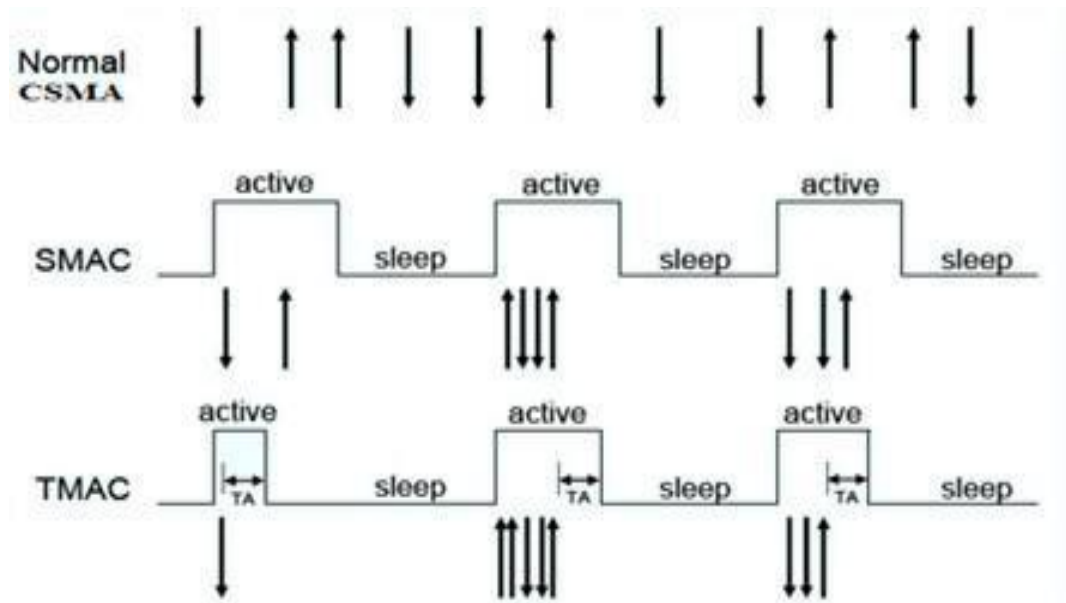


Рис. 3.1. Цикли сну та пробудження в *SMAC* та *TMAC*

Синхронізація часу потрібна, щоб одержувач залишався активним, коли відправник надсилає повідомлення. У періоді синхронізації часу першим кроком у

встановленні розкладу сну для вузла є передача пакету *SYNC* від сусіда. Пакет *SYNC* містить графік сну та вказує на те, що відправник переходить у сплячий режим через t секунд. Як тільки вузол отримує графік сну свого сусіда, він приймає цей графік і повторно передає графік для інших сусідніх вузлів для прийняття. Якщо вузол не отримує пакет *SYNC* протягом заздалегідь визначеного періоду очікування, вузол встановлює і транслює власний розклад. Прикордонні вузли (вузли між двома активними розкладами) можуть отримувати два різні розклади від різних вузлів. Вони можуть прийняти обидва або один із графіків. Були застосовані *SMAC*, де прикордонні вузли відповідають обом графікам. Завдяки цьому механізму мережа ділиться на кілька віртуальних кластерів, кожен кластер оточений прикордонними вузлами. Кожен вузол у кластері дотримується однакового розкладу сну тоді як прикордонні вузли відповідають графіку обох сусідніх кластерів. Отже, прикордонні вузли залишаються пробудженими протягом більшого періоду, збільшуючи тим самим споживання енергії. Однак у кожному циклі кордонні вузли та віртуальні кластери постійно змінюються. Отже, віртуальна кластеризація не впливає на термін служби мережі в цілому.

У *SMAC* обмін даними відбувається в період передачі даних через *RTS-CTS-DATA-ACK* для одноадресного зв'язку. Вузол може продовжити свою активну тривалість, якщо обмін даними не закінчується в активний період. Однак, навіть якщо обмін даними закінчується протягом активного періоду, вузол все одно залишатиметься активним до часу свого сну, витрачаючи тим самим енергію. Коли вузол надсилає *RTS* або *CTS*, він позначає тривалість передачі даних у пакетах *RTS* або *CTS*. Сусідні вузли, які підслуховують ці *RTS* або *CTS*, встановлюють *NAV* залежно від тривалості і переходять у режим сну, оскільки вони не можуть спілкуватися в цій тривалості через порушення сусідів. Коли *NAV* перериває пожежу, ці сусідні вузли знову починають слідувати своєму звичайному графіку. Цей прийом рукостискання не тільки зменшує зіткнення, але й економить енергію, підслуховуючи уникнення. Рис. 3.2 демонструє це спостереження, оскільки споживання енергії є низьким при збільшенні потоку та збільшується із збільшенням потоку. Це пов'язано з тим, що сусідні вузли довше сплять на

підслуховуванні *RTS* або *CTS*, частота яких вища при великому трафіку. Однак такого збереження не вдається досягти в широкомовних повідомленнях, оскільки мовлення не використовує *RTS-CTS*.

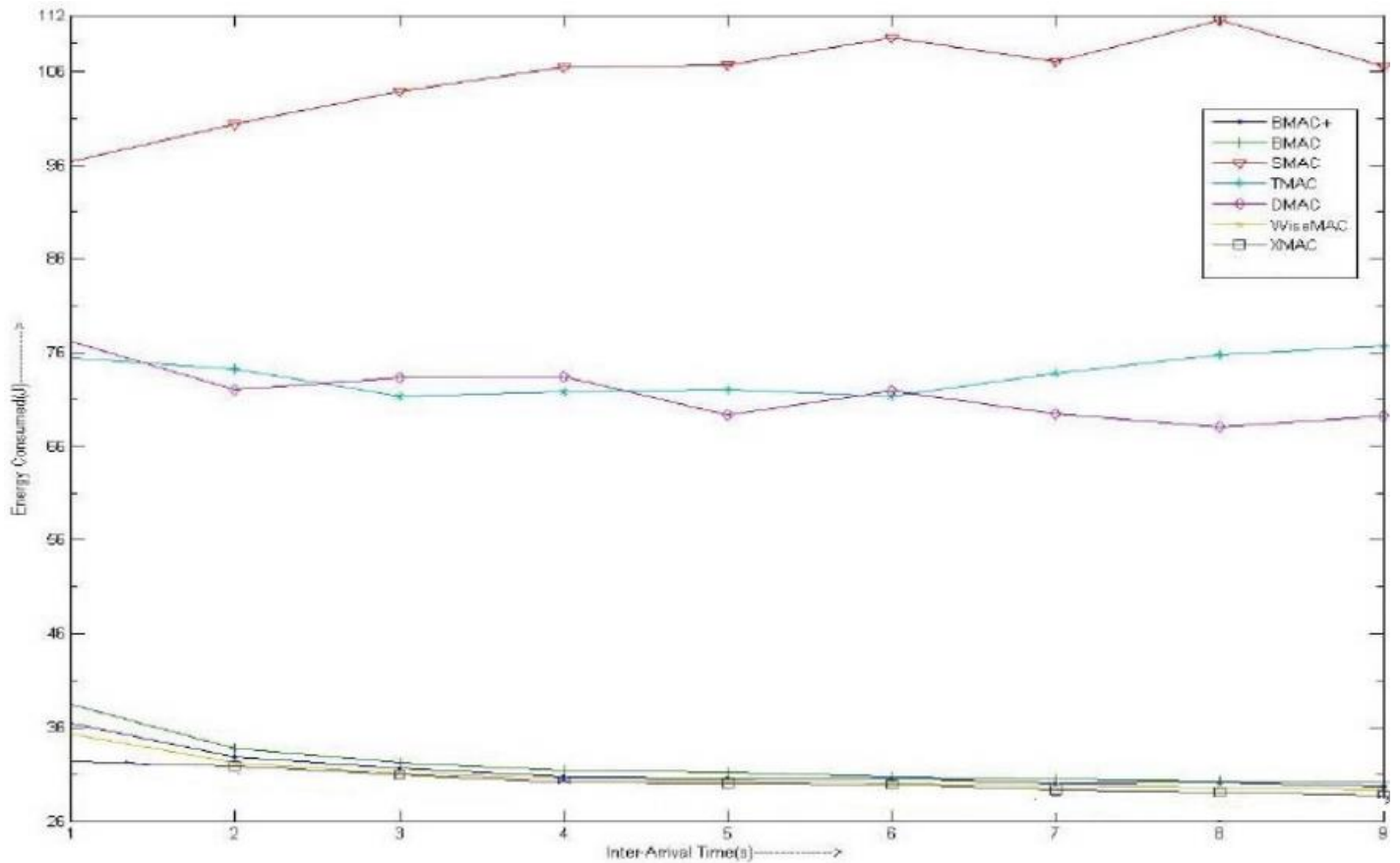


Рис.3.2. Споживання енергії у порівнянні з часом міжприбуття для конвергентного зв'язку

3.3.2. T-MAC

Тайм-аут *MAC* (*TMAC*) [68] також є протоколом рівня *MAC* на основі суперечок, який базується на основних особливостях *SMAC* в оптимізації енергоефективності шляхом сну під час періодичної бездіяльності мережі. Однак, на відміну від *SMAC*, *TMAC* дотримується динамічного розкладу сну. Протокол *TMAC* вводить активний механізм очікування, який зменшує накладні витрати на прослуховування в режимі

очікування, динамічно регулюючи активний період відповідно до навантажень мережевого трафіку. *ТМАС* дозволяє вузлам спати через деякий час, коли весь мережевий трафік буде завершено, як це пояснено на рис 3.1. Кінець трафіку сигналізується після моніторингу простою каналу на період адаптивного тайм-ауту (ТА). Якщо протягом цього періоду часу ТА ніякої активності не відбувається, вузол вимикає радіо і переходить у режим сну. Період ТА повинен бути достатньо великим, щоб подолати проблему раннього сну (вузол переходить у стан сну, коли сусід все ще має відправити пакети). Така процедура робить *ТМАС* більш енергійним, ніж *SMAC*, як видно з рис. 3.2.

Тривалість ТА залежить від інтервалу суперечок, тривалості пакета *RTS* та часу обробки. Ми знаходимо відповідний час ТА для нашої реалізації, як показано на рис 3.3.

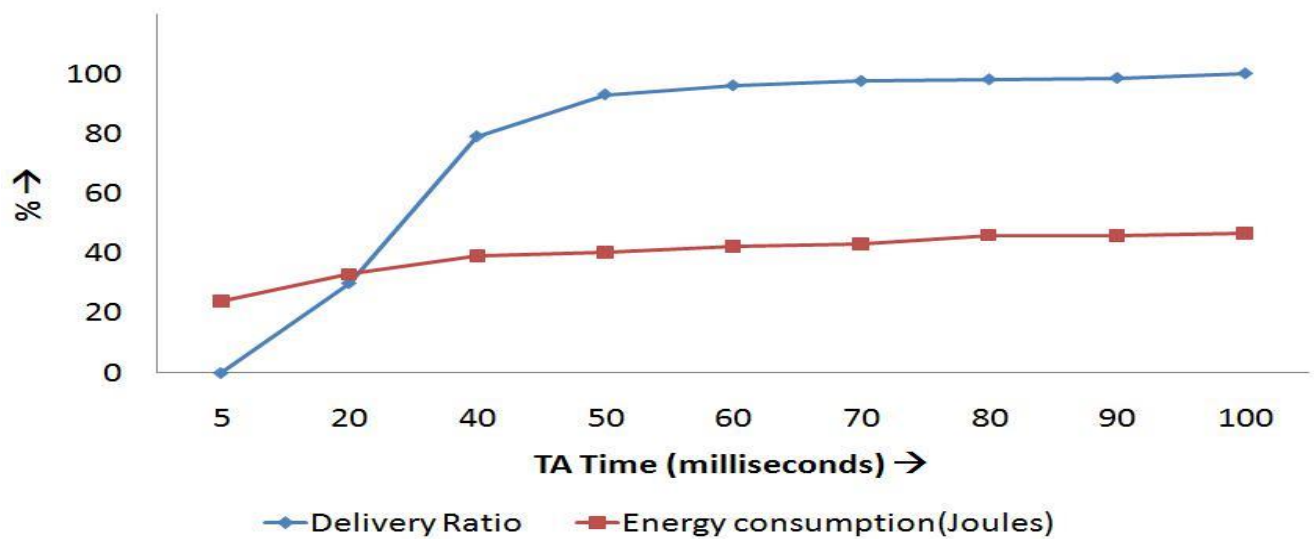


Рис. 3.3. Вплив зміни часу ТА на коефіцієнт доставки та споживання енергії

На графіку представлений аналіз компромісу між спожитою енергією та коефіцієнтом доставки, оскільки ТА варіюється. Графік ненульового перетину показує, що витрачається значна кількість енергії, навіть коли немає трафіку даних. Ця енергія споживається протягом періоду синхронізації часу. Отже, синхронізація - це великі накладні витрати в *SMAC* та *ТМАС*.

TMAC також вводить механізм *FRTS* та повний пріоритет буфера, щоб уникнути проблем раннього сну для конвергентного типу передачі даних. Коли вузол, який має дані для відправки, підслуховує пакет *CTS*, він транслює *FRTS*. Тривалість даних зберігається у *FRTS*-пакетах. Одержувач *FRTS* встановлює свій *NAV* і лягає спати. Після зв'язку, вузол знову прокидається, щоб отримати дані відправника *FRTS*. При повному пріоритеті буфера, коли буфер надсилання вузла заповнений і він отримує *RTS* від якогось вузла, тоді замість відповіді *CTS* вузол передає власний *RTS*, таким чином приймаючи пріоритет. Після завершення надсилання даних лише тоді він відповідає *CTS* на вихідний запит *RTS*, який він отримав. Отже, цей механізм також вводить контроль потоку в потік даних. Слід зазначити, що повний пріоритет буфера слід застосовувати лише в конвергентному типі механізму, а не в спеціальному типі зв'язку.

3.3.3. *D-MAC*

D-MAC [68] - це протокол, який спрямований на доставку даних у режимі реального часу, але при цьому залишається енергоефективним. Він приймає розподілений шаблон пробудження для переадресації пакетів даних на базову станцію, як показано на рис. 3.4. Вузли вважаються присутніми на різних рівнях дерева збору даних.

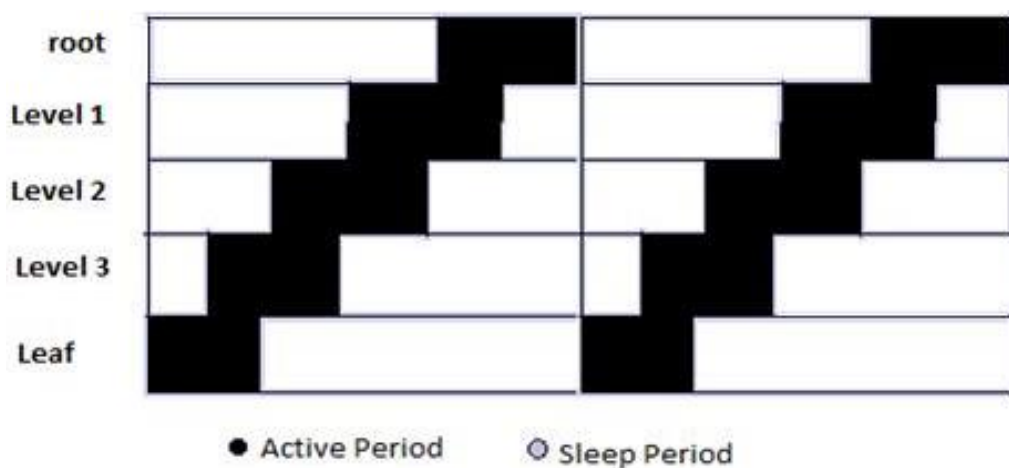


Рис. 3.4. Активний та період сну *DMAC*

Всі вузли на одному рівні пробуджуються одночасно, щоб отримувати дані. Цей період прийому, μ , супроводжується періодом передачі (μ), в якому вони пересилають дані на вищий рівень. Вузли на наступному рівні прокидаються відразу після періоду прийому нижчого рівня. Отже, активний період - це розподілений шаблон пробудження, де активний період одного рівня частково перекривається з періодом нижчого рівня, як показано на рис. 3.4. Завдяки такому розподіленому шаблону пробудження, пакет даних сягає від кореня до листя в один лише цикл, тим самим мінімізуючи затримку. *D-MAC* застосовує метод передбачення даних, коли декільком спадкоємцям потрібно надсилати дані одному з надкласів лише за один цикл. При передбаченні даних, якщо надклас отримує дані, він знову прокидається через 3μ , сподіваючись, що будуть дані і від іншої дитини.

Незважаючи на використання прогнозування даних та прапора *MTS*, *D-MAC* не підходить для великого навантаження через невеликий час μ . *D-MAC* не використовує *RTS-CTS*, оскільки в даний момент часу лише декілька вузлів мережі залишатимуться активними, тим самим зменшуючи ймовірність зіткнення. Крім того, агрегування даних можливо на кожному вузлу, оскільки надклас може отримувати пакети від усіх дочірніх об'єктів перед їх переадресацією. *D-MAC* вимагає локальної та ефективної синхронізації. Через розподілений графік пробудження кожен вузол повинен знати свій рівень глибини. Було реалізовано *D-MAC*, коли пакет синхронізації опускається від кореневих до листових вузлів, інформуючи кожен вузол про рівень їх глибини. Іншим недоліком є те, що *D-MAC* не може використовуватися для спілкування за місцевими плітками через його ієрархічну конструкцію пересилання даних.

3.3.4. *B-MAC*

BMAC [69] - це протокол рівня *MAC* для бездротових сенсорних мереж, який використовує адаптивну схему вибірки преамбули. Ця методика полягає у відборі проб середовища з фіксованими інтервалами часу. Рис. 3.5 описує роботу *BMAC*. Вибірка

середовища означає прослуховування каналу для певної діяльності. У цій схемі кожен вузол відбирає середовище через фіксовані інтервали, щоб перевірити, чи бажає будь-який вузол спілкуватися. Якщо який-небудь вузол має пакет для відправки, вузол (відправник) відчуває носій, якщо він вільний, робить невеликий відступ, а потім надсилає довгу преамбулу пробудження, за якою йде пакет даних. Преамбула - це не пакет, а RF-імпульс фізичного рівня, який просто перевищує тривалість періоду дискретизації, так що вузол дискретизації середовища помічає цю активність. Коли приймач прокидається, він відчуває середовище, і якщо виявляє будь-який шум (преамбула), він включає радіо і чекає закінчення преамбули. Після завершення преамбули, якщо пакет даних призначений самому вузлу, він отримує повний пакет даних, інакше ігнорує пакет і переходить у режим сну.

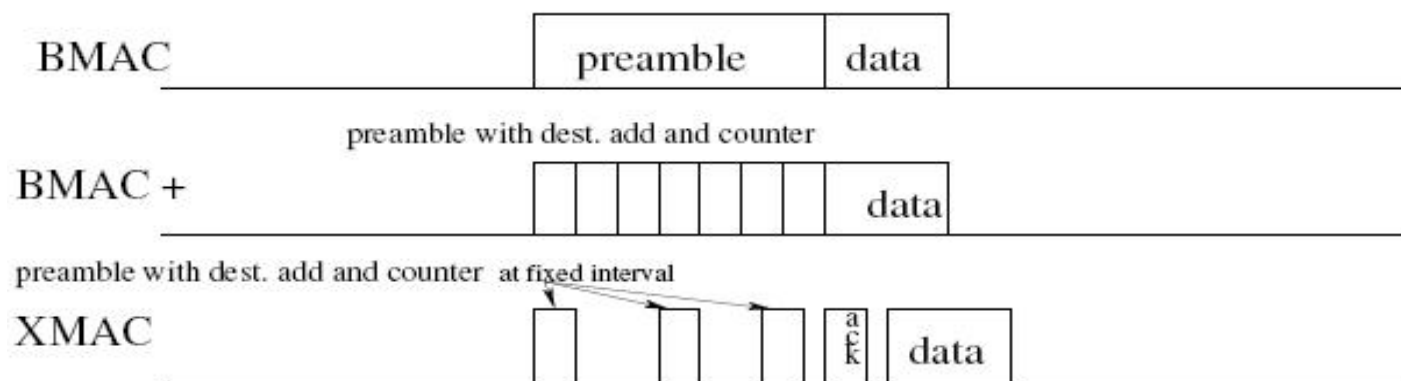


Рис. 3.5. Преамбула в *BMAC*, *BMAC +* та *XMAC*

Цілями протоколу *BMAC* є робота з низьким енергоспоживанням, ефективне уникнення зіткнень та ефективне використання каналів при низькій, а також високій швидкості передачі даних. *BMAC* можна масштабувати до великої мережі. Він переконфігурується мережами, його реалізація проста і вимагає невеликого обсягу оперативної пам'яті. Протокол *BMAC* використовує концепції функціональності доступу до медіа. Він використовує чітку оцінку каналів (*CCA*) та зворотне відключення для арбітражу каналу, підтвердження надійності та прослуховування з низьким енергоспоживанням (*LPL*) для зв'язку низької потужності. *B-MAC* - це лише протокол

зв'язку, з мережевими послугами, такими як організація, синхронізація та маршрутизація, побудовані над його реалізацією, але *ВМАС* не в змозі забезпечити багатопакетні механізми, такі як підтримка прихованих терміналів, фрагментація повідомлень та особлива політика щодо низького енергоспоживання.

3.3.5. *В-МАС* +

ВМАС + [70] є розширенням протоколу *ВМАС* [69]. *ВМАС* + намагається зменшити витрату енергії завдяки довгій преамбулі *ВМАС*. Преамбула - це послідовність бітів, яка не містить жодної відповідної інформації і використовується для того, щоб повідомляти приймачу, що якийсь вузол хоче спілкуватися. Основна ідея *ВМАС* + полягає в тому, щоб замінити преамбулу пробудження невеликою кількістю блоків, що містять деяку інформацію, як показано на рис. 3.5. Ця інформація містить адресу вузла призначення та кількість решти блоків або зворотний відлік даних, починаючи з найбільшого числа відповідно до необхідної довжини преамбули. . Адреса призначення використовується для уникнення підслуховування без отримання решти блоків преамбули. Кількість блоків, що залишилися, або відлік номера послідовності, починаючи з нуля, використовується, щоб уникнути прослуховування в режимі очікування вузлів, які не є одержувачами даних. *ВМАС* + економить більше енергії, ніж *ВМАС*, з однаковою затримкою та пропускнуою здатністю, як це видно з результатів моделювання на рис. 3.6. Це пояснюється тим, що в *ВМАС* +, коли приймач отримує ранню преамбулу, він може вимкнути радіостанцію, щоб залишилися блоки преамбули, і дочекатися, поки ввімкнеться час надходження даних увімкнене радіо. Він знову прокинеться, коли надійдуть дані.

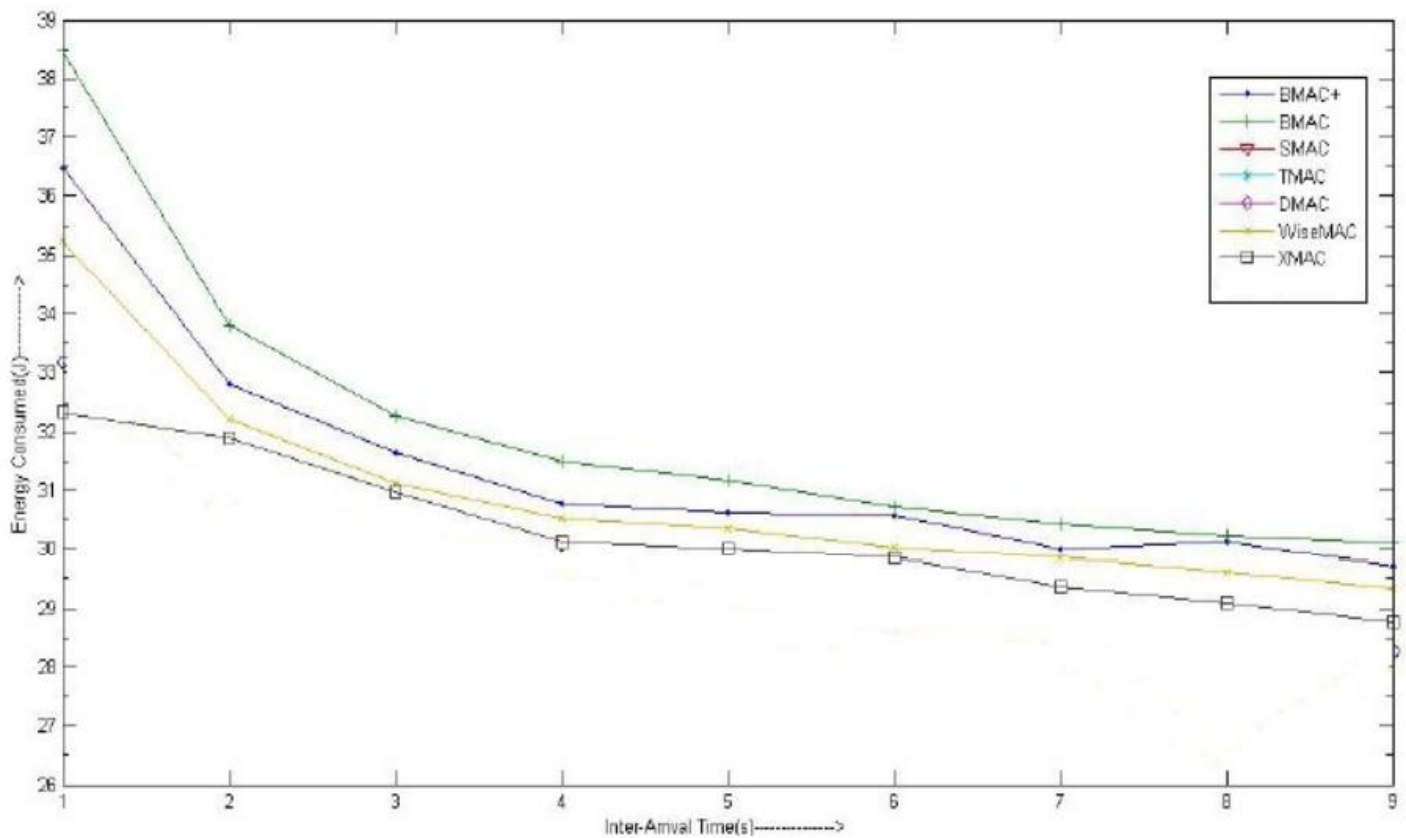


Рис. 3.6. Ближчий вигляд аналізу на рис. 3.2

3.3.6. X-MAC

Стандартні протоколи MAC, такі як BMAC, використовують довгу преамбулу перед даними для пробудження приймача. Ця схема преамбули була вдосконалена в BMAC + для зменшення споживання енергії. У приймачі BMAC вимикає радіоприймач після отримання преамбули, але відправник продовжує надсилати решту частини преамбули, що призводить до втрати енергії, а також призводить до надмірної затримки.

У 2006 році ці проблеми були вирішені, коли був представлений XMAC [71], протокол MAC низької потужності. XMAC запроваджує скорочений підхід до преамбули, який зберігає переваги прослуховування з низьким енергоспоживанням, таке як зв'язок із низьким енергоспоживанням, простота та роз'єднання графіків сну передавача-приймача. XMAC представляє серію коротких пакетів преамбули, кожен

пакет містить адресу призначення та залишкову кількість преамбул. Вузол надсилає ці серії преамбул з фіксованими інтервалами, як показано на рис. 3.5. Цей інтервал достатньо довгий, щоб отримати відповідь від приймача. Коли одержувач отримує будь-яку преамбулу, він відразу надсилає підтвердження відправнику протягом фіксованого інтервалу між пакетами преамбули. Коли відправник отримав підтвердження, він припиняє надсилання подальших преамбул і негайно надсилає пакет даних. Це зменшує енергію як з боку приймача, так і з боку відправника, а також зменшує затримку на стрибок.

Крім того, коли відправник, який чекає чіткого каналу для відправки даних, виявляє преамбулу, а потім чує підтвердження від вузла, на який сам відправник хоче надіслати дані, він приймає випадковий зворотний бік, достатній для завершення передачі даних триває в даний час. Після завершення цієї передачі він надсилає дані безпосередньо без будь-якої преамбули. Рандомізоване відкликання необхідне, щоб уникнути зіткнення, якщо більше одного вузла намагається надіслати дані одночасно. Оскільки цей метод вимагає, щоб вузол приймача залишався активним, щоб отримувати дані також у наступному циклі, кожен приймач у *XMAC* залишається активним протягом короткого періоду, якщо є додаткові вузли, які бажають відправити дані. Ці два способи зменшують велике споживання енергії, а також зменшують час затримки.

3.3.7. *WiseMAC*

WiseMAC [72] - це протокол управління доступом для *WSN*, який базується на непостійному *CSMA* і використовує техніку дискретизації преамбули для зменшення енергоспоживання. *WiseMAC* намагається використовувати мінімальну преамбулу пробудження. *WiseMAC* не вимагає налаштування сигналізації, ніякої загальномережної синхронізації і адаптивний до навантаження трафіку. *WiseMAC*, як *BMAC*, базується на техніці дискретизації, коли вузол прослуховує канал протягом короткого часу. Усі вузли датчика сприймають середовище в один і той же постійний період '*TW*' незалежно. Тут

"незалежний" означає, що вони можуть відчувати середовище в різний час, але вони беруть вибірку середовища для того самого періоду. У *ВМАС*, якщо носій виявляється зайнятим, вузол постійно прослуховує носій до отримання даних або до тих пір, поки носій знову не стане простоєм. На стороні відправника, преамбула пробудження розміром, рівним періоду дискретизації, додається перед кожним кадром даних, щоб одержувач прокинувся під час надходження пакету даних. Цей протокол забезпечує найкращий результат, коли носій не працює, але недоліком цього протоколу є те, що преамбули тривалого пробудження спричиняють обмеження пропускну здатності та великі витрати енергії на передачу та прийом. Основна ідея *WiseMAC* - вивчення графіків вибірки безпосередніх сусідів вузла. Ці графіки використовуються для мінімізації розміру преамбул. Для відновлення втрат пакетів у *WiseMAC* використовується підтвердження рівня зв'язку. Пакети *WiseMAC ACK* використовуються не тільки для передачі інформації про підтвердження, але й для інформування інших вузлів (включаючи відправника) про час, що залишився до наступної вибірки. Ці інші вузли зберігають цього разу у своїх таблицях. Використовуючи цю інформацію, вузол передає пакет із мінімізованим розміром преамбули. Тривалість преамбули пробудження охоплює невеликий потенційний зсув годинника між годинником у джерела та пункту призначення. *WiseMAC* використовує наступну рівність для обчислення мінімальної преамбули. $TP = \min(4\theta L, TW)$, де θ - допуск на частоту кварцу часової бази, а L інтервал між зв'язками. L також оновлюється, коли вузол отримує *ACK* своїх сусідів на рівні *Link*. Перший зв'язок між двома вузлами завжди здійснюється з використанням тривалої пробудження, що дорівнює TW . Як тільки якась інформація про час буде знайдена, використовується преамбула пробудження зменшеного розміру. Оскільки преамбула пропорційна інтервалу L між комунікаціями, вона стане малою, коли трафік високий. Це робить *WiseMAC* адаптивним до трафіку. Накладні витрати на пакет зменшуються із збільшенням трафіку. В умовах низького трафіку накладні витрати на пакет високі, але середнє споживання енергії завдяки цим накладним витратам є низьким. Інша важлива річ щодо *WiseMAC* полягає в

тому, що він використовує більше бітів, присутніх у заголовку пакета даних, як протокол енергозбереження *IEEE 802.11*. Якщо для цього біта встановлено значення 1, це означає, що для того самого вузла надходить більше даних. Отже, приймач продовжує перевіряти носій навіть після надсилання підтвердження. Ми змоделювали *WiseMAC* в *PowerTossim-Z*, а результати моделювання показані на рис. 3.6. На рисунку ми бачимо, що споживання енергії в *WiseMAC* менше, ніж будь-який інший протокол *MAC*, але більше, ніж *XMAC*. Причиною цього є те, що, хоча *XMAC* не використовує мінімізовану преамбулу, але в *XMAC* всі сусідні вузли вимикають своє радіо після отримання будь-якої окремої преамбули, перебуваючи у *WiseMAC*,

Таким чином, енергія відправника та одержувача може бути збережена, але включаючи загальну енергію всіх сусідніх вузлів більше, ніж енергія *XMAC*. Однак *WiseMAC* працює краще, ніж інші, і може бути більш корисним в умовах низького трафіку.

3.4. Розкладання системи

У звичайних системах більша частина стеку мережевих протоколів реалізована на основному процесорі. Таким чином, мережевий інтерфейс та основний процесор завжди повинні бути увімкненими, щоб мережа була активною. Оскільки майже всі дані транспортуються через процесор, продуктивність та споживання енергії є суттєвою проблемою.

У системі зв'язку місцевість посилення може бути використана шляхом декомпозиції стеку мережевих протоколів та обережного управління потоком даних. Це може зменшити споживання енергії з кількох причин.

По-перше, коли система побудована з незалежних компонентів, які реалізують різні рівні стека зв'язку, непотрібні копії даних між послідовними шарами стеку протоколів можуть бути усунені. Це виключає марнотратну передачу даних по

глобальній шині і, таким чином, економить великі витрати в автобусах, мультиплексах та драйверах.

По-друге, спеціальне обладнання може виконувати базову обробку сигналу і може переміщувати необхідні дані безпосередньо до місця призначення, тим самим зберігаючи копії даних на системній шині. Більше того, це спеціальне обладнання може виконувати свої завдання набагато ефективніше, ніж загальний процесор.

Нарешті, комунікаційний процесор може бути застосований для обробки більшості нижчих рівнів стеку протоколів, тим самим дозволяючи основному процесору спати протягом тривалих періодів часу, не впливаючи на продуктивність або функціональність системи.

Це розкладання також може бути застосовано поза системним рівнем портативної системи: певні функції системи можуть бути перенесені з портативної системи на віддалений сервер, який має велику кількість енергетичних ресурсів. Цей віддалений сервер обробляє ті функції, які неможливо ефективно обробляти на портативній машині. Наприклад, базова станція може обробляти частини стека мережевих протоколів замість мобільного. Віддалений сервер має приватний виділений протокол зв'язку з мобільним, так що мобільні блоки можуть використовувати внутрішній, легкий, протокол для зв'язку з базовою станцією, а не *TCP / IP* або *UDP*. Кінцевий результат - економія коду та енергії.

3.5. Мережі короткого радіусу дії низької потужності

Портативні комп'ютери повинні мати можливість безперешкодного переходу від одного комунікаційного середовища до іншого, наприклад, від мережі *GSM* до внутрішньої мережі, без перезавантаження або перезапуску програм. Програми вимагають, щоб мережі могли визначати, що мобільний телефон перейшов в можливі *QoS*.

З однієї мережі в іншу з можливою різною якістю обслуговування. Мережа, яка є найбільш підходящою в певному місці в певний час, залежить від вимог користувача, пропускної здатності мережі, витрат на зв'язок, споживання енергії тощо. Система та додатки можуть адаптуватися до вартості зв'язку (наприклад, вимірюється в ампер-годин або телефонних рахунків).

На коротких відстанях, як правило, до п'яти метрів, можливий високошвидкісний, низькоенергетичний зв'язок[62]

Приватні будинки, офісні будівлі та громадські будівлі можуть бути оснащені мікро-стільниковими мережами з невеликою антеною у кожному приміщенні регулярно проміжки часу, так що мобільний комп'ютер ніколи не повинен спілкуватися на великій відстані - таким чином економить енергію - і таким чином, що пропускна здатність, доступна в ефірі, не повинна ділитися з великою кількістю інших пристроїв - таким чином забезпечуючи високу сукупність пропускної здатності. На великих відстанях мобільний телефон може використовувати стандартну інфраструктуру цифрової телефонії (наприклад, GSM).

Висновки за розділом

У цьому підрозділі ми обговорили деякі методи, які можуть бути використані для зменшення споживання енергії, необхідної для (бездротового) зв'язку поза комп'ютером. Взагалі кажучи, мінімізація споживання енергії є завданням, яке вимагає мінімізації внесків зв'язку та обчислень, внесення відповідних компромісів між ними. Наприклад, зменшення обсягу переданих даних може бути корисним. З іншого боку, вартість обчислення (наприклад, для стиснення даних, що надсилаються) може бути високою, і в крайньому випадку вона може бути такою, що було б краще просто надіслати необроблені дані.

Сучасні високопродуктивні мережеві протоколи вимагають, щоб весь доступ до мережі здійснювався через операційну систему, що додає значні накладні витрати як на

шлях передачі (як правило, системний виклик і копіювання даних), так і на шлях прийому (зазвичай переривання, системний дзвінок і копія даних). Це не тільки спричиняє проблеми з роботою, але й значне споживання енергії. Розумні мережеві інтерфейси можуть деяким чином усунути цю проблему. Для вирішення проблеми продуктивності були розроблені кілька архітектур комунікації на рівні користувача, які вилучають операційну систему з критичного шляху зв'язку.

Ми порівняли протоколи *MAC* на основі *CSMA* щодо їх споживання енергії та виявили, що в цілому *X-MAC* працює краще, ніж усі інші розглянуті протоколи.

ВИСНОВКИ

Потреба в надійному моделюванні та оптимізації енергоефективності на всіх системних рівнях буде продовжувати зростати з урахуванням завтрашніх навантажень та вимог щодо продуктивності як для систем низького, так і високого класу. Такі моделі, забезпечуючи оптимізацію часу проектування або часу роботи, дозволять дизайнерам зробити правильний вибір при визначенні майбутнього покоління енергоефективних систем.

У роботі обговорювалися різні ідеї та методи, запропоновані в літературі, з метою розробки енергосвідомих комп'ютерних систем. Різні обстежені методи відрізнялись між собою аналізом потужності проектування та методами динамічного управління енергією під час виконання, а також були охоплені методи, які можуть бути використані для зменшення споживання енергії, необхідної для (бездротового) зв'язку поза комп'ютером.

Методи аналізу потужності в основному засновані на моделюванні. Ці методи інтегрують різні енергетичні моделі в існуючі інструменти моделювання та аналізують та профілюють споживання енергії на різних рівнях комп'ютерної системи під час проектування, щоб допомогти побудувати енергоефективні апаратні та програмні системи.

Споживана центральним процесором енергія є основною частиною загального енергоспоживання комп'ютерної системи і, отже, стала основною метою аналізу енергоспоживання. Кілька моделей живлення були розроблені та інтегровані в існуючі симулятори продуктивності, щоб дослідити споживання енергії центрального процесора як на основі функціональних блоків, так і на процесорі в цілому.

Однак необхідно було врахувати й інші критичні компоненти, щоб зменшити загальну енергію системи. Були розглянуті апаратні моделі на рівні стану, де загальне споживання енергії всієї системи оцінюється на основі стану кожного пристрою, що допомагає продовжити життя системи якомога довше. Представлені програмні підходи,

що визначають точки доступу до енергії в додатках та процедурах операційної системи, і таким чином дозволяють програмістам видаляти вузькі місця або модифікувати програмне забезпечення таким чином, щоб воно було енергоефективним. Представлений повний інструмент моделювання на системному рівні, який моделює апаратні компоненти, такі як ЦП, ієрархія пам'яті та підсистема диска з низьким енергоспоживанням, а також програмні компоненти, такі як ОС та додатки.

Після представлення енергетичних моделей на рівні процесора та системному рівні, були описані енергетичні моделі на паралельному системному рівні з акцентом на мережах взаємозв'язку. З постійно зростаючим попитом на обчислювальну потужність процесори стають дедалі більш взаємопов'язаними, створюючи великі кластери комп'ютерів, що обмінюються даними мереж, що під'єднані до інших мереж. Аналіз потужності в цій області націлений на будівельні блоки всередині мережевого маршрутизатора та комутаційної тканини.

З іншого боку, під час роботи застосовуються динамічні методи управління енергією. Вони відстежують навантаження системи, щоб передбачити майбутні вимоги до обчислень і намагаються динамічно адаптувати поведінку системи відповідно. Методи *DPM* націлені на зменшення споживання енергії під час роботи шляхом вибіркового вимкнення або уповільнення компонентів, коли системи простоюють або обслуговують невеликі робочі навантаження. Методи *DPM* застосовуються по-різному та на різних рівнях.

Були розглянуті методи *DPM*, класифіковані на основі рівня впровадження:

- на рівні ЦП;
- на системному рівні, що розглядають інші системні компоненти (пам'ять, жорсткий диск, пристрої вводу-виводу, дисплей тощо), ніж ЦП;
- методи *DPM*, запропоновані для паралельних систем, де кілька вузлів співпрацюють, щоб заощадити загальну потужність, колективно виконуючи задане паралельне завдання.

Успішне проектування та оцінка методів управління енергією та оптимізації сильно залежить від наявності широкого та точного набору інструментів аналізу потужності, які буде вирішальним у будь-якому майбутньому дослідженні. Хоча акцент повинен бути більше на загальній поведінці системи

Ми обговорили деякі методи, які можуть бути використані для зменшення споживання енергії, необхідної для (бездротового) зв'язку поза комп'ютером. Взагалі кажучи, мінімізація споживання енергії є завданням, яке вимагає мінімізації внесків зв'язку та обчислень, внесення відповідних компромісів між ними. Наприклад, зменшення обсягу переданих даних може бути корисним. З іншого боку, вартість обчислення (наприклад, для стиснення даних, що надсилаються) може бути високою, і в крайньому випадку вона може бути такою, що було б краще просто надіслати необроблені дані.

Сучасні високопродуктивні мережеві протоколи вимагають, щоб весь доступ до мережі здійснювався через операційну систему, що додає значні накладні витрати як на шлях передачі (як правило, системний виклик і копіювання даних), так і на шлях прийому (зазвичай переривання, системний дзвінок і копія даних). Це не тільки спричиняє проблеми з роботою, але й значне споживання енергії. Розумні мережеві інтерфейси можуть деяким чином усунути цю проблему. Для вирішення проблеми продуктивності були розроблені кілька архітектур комунікації на рівні користувача, які вилучають операційну систему з критичного шляху зв'язку.

Ми оцінили всі вищеописані протоколи *MAC* за параметрами. Після впровадження протоколи були інтегровані з *PowerTOSSIM-Z* для вимірювання споживання енергії та виявили, що в цілому *X-MAC* працює краще, ніж усі інші розглянуті протоколи.

Тоді як енергоефективна конструкція системи важлива для бюджетних підприємств портативних систем, це також вірно для паралельних та кластерних систем високого класу. Це тому, що високий рівень споживання енергії підвищує температуру і тим самим погіршує продуктивність та надійність. Отже, цілісний підхід, який враховує всі компоненти таких систем, повинен бути подальше досліджуваним.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A. Azevedo, R. Cornea, I. Issenin, R. Gupta, N. Dutt, A. Nicolau, and A. Veidenbaum, "Architectural and Compiler Strategies for Dynamic Power Management in the COPPER Project," *IWIA 2001 International Workshop on Innovative Architecture*, 2001.
2. A. Azevedo, R. Cornea, I. Issenin, R. Gupta, N. Dutt, A. Nicolau, and A. Veidenbaum, "Profile-based Dynamic Voltage Scheduling using Program Checkpoints," *Design, Automation and Test in Europe Conference and Exhibition (DATE'02)*, 2002.
3. A. Bavier, A. Montz and L. Peterson, "Predicting MPEG Execution Times," *Proceedings of SIGMETRICS '98/PERFORMANCE '98*, 1998.
4. L. Benini, A. Boglio, and G. De Micheli, "A Survey of Design Techniques for System-level Dynamic Power Management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Volume 8, Issue 3, June 2000.
5. L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli, "Policy Optimization for Dynamic Power Management," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 18, No. 6, pp. 813–833, June 1999.
6. D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta and P. Cook, "Power Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors," *IEEE Micro*, pp. 26-44, December 2000.
7. D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," *Proceedings of the 27th International Symposium on Computer Architecture (ISCA)*, June 2000.
8. D. Burger and T. Austin, "The SimpleScalar Tool Set, Version 2," *Tech. Report No. 1342, Computer Sciences Dept. Univ. of Wisconsin*, June 1997.
9. F. Chang, K. Farkas and P. Ranganathan, "Energy-driven Statistical Profiling: Detecting Software Hotspots," *Proceedings of the Workshop on Power Aware Computing Systems*, February 2002.

10. W. Chedid and C. Yu, "Dynamic Voltage Scaling Techniques for Power-Aware MPEG Decoding", *Master's Thesis, ECE Dept., Cleveland State University, December 2003.*
11. E.-Y. Chung, L. Benini, A. Bogliolo, Y.-H. Lu, and G. De Micheli, "Dynamic Power Management for Nonstationary Service Requests," *IEEE Trans. on Computers, Vol. 51, No. 11, pp. 345–1361, November 2002.*
12. E.-Y. Chung, L. Benini, and G. De Micheli, "Dynamic Power Management Using Adaptive Learning Tree," *Proceedings of the International Conference on Computer-Aided Design, pp. 274–279, November 1999.*
13. T. Cignetti, K. Komarov and C. Ellis, "Energy Estimation Tools for the PalmTM," *Proceedings of the ACM MSWiM'2000: Modeling, Analysis and Simulation of Wireless and Mobile Systems, August 2000.*
14. F. Douglass, P. Krishnan, and B. Bershad, "Adaptive Disk Spin-down Policies for Mobile Computers," *Proceedings 2nd USENIX Symp. on Mobile and Location-Independent Computing, pp. 381–413, 1995.*
15. E. N. Elnozahy, M. Kistler and R. Rajamony, "Energy-Efficient Server Clusters," *Proceedings of the Second Workshop on Power Aware Computing Systems, February 2002.*
16. J. Flinn and M. Satyanarayanan. "PowerScope: A tool for Profiling the Energy Usage of Mobile Applications," *Proceedings of the Workshop on Mobile Computing Systems and Applications (WMCSA), February 1999.*
17. S. Gary, P. Ippolito, G. Gerosa, C. Dietz, J. Eno, and H. Sanchez, "PowerPC 603, A Microprocessor for Portable Computers," *IEEE Design & Test of Computers, Volume 11, Issue 4, pp. 14-23, October 1994.*
18. N. A. Ghazaleh, D. Mosse, B. Childers, R. Melhem, and M. Craven, "Collaborative Operating System and Compiler Power Management for Real-Time Applications," *The 9th IEEE Real-Time and Embedded Technology and Applications Symposium, 2003.*
19. C. Gniady, Y. C. Hu, Y.-H. Lu, "Program Counter Based Techniques for Dynamic Power Management," *10th International Symposium on High Performance Computer Architecture (HPCA'04), February 2004.*

20. R. Golding, P. Bosch, C. Staelin, T. Sullivan, and J. Wilkes, "Idleness is Not Sloth," *Proceedings of the USENIX Winter Conference*, pp. 201–212, 1995.
21. K. Govil, E. Chan and H. Wasserman, "Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU," *MobiCom 1995*.
22. M. Gowan, L. Biro, and D. Jackson, "Power Considerations in the Design of the Alpha 21 264 microprocessor," *ACM Design Automation Conference*, pp. 726-731, June 1998.
23. S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, and M. Kandemir, "Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach," *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA-8)*, February 2002.
24. C.-H. Hwang and A. Wu, "A Predictive System Shutdown Method for Energy Saving of Event-driven Computation," *International Conference on Computer-Aided Design*, pp. 28–32, November 1997.
25. J. Henkel and H. Lekatsas, "A2BC: Adaptive Address Bus Coding for Low Power Deep SubMicron Designs," *ACM Design Automation Conference*, 2001.
26. M. Kandemir, N. Vijaykrishnan, M. Irwin, "Compiler Optimizations for Low Power Systems", *Workshop on Power Aware Computing Systems*, pp. 191-210, 2002.
27. E. J. Kim, K. H. Yum, G. M. Link, C. R. Das, N. Vijaykrishnan, M. Kandemir and M. J. Irwin, "Energy Optimization Techniques in Cluster Interconnects", *International Symposium on Low Power Electronics and Design (ISLPED'03)*, August, 2003.
28. J. Li, J. F. Martíne, and M. C. Huang, "The Thrifty Barrier: Energy-Aware Synchronization in Shared-Memory Multiprocessors," *10th International Symposium on High Performance Computer Architecture (HPCA'04)*, February 2004.
29. Y.-H. Lu, L. Benini, and G. De Micheli, "Low-Power Task Scheduling for Multiple Devices," *International Workshop on Hardware/Software Codesign*, May 2000.
30. Y.-H. Lu, L. Benini, and G. De Micheli, "Operating-System Directed Power Reduction", *International Symposium on Low Power Electronics and Design*, July 2000.

31. Y.-H. Lu and G. De Micheli, "Comparing System-Level Power Management Policies," *IEEE Design & Test of Computers*, Volume 18, Issue 2, pp. 10-19, March 2001.
32. Y.-H. Lu, T. Simunic, and G. De Micheli, "Software Controlled Power Management," *Proceedings of the 7th International Workshop on Hardware/Software Codesign (CODES99)*, pages 157–161, May 1999.
33. C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley Publishing Company, Inc., 1980.
34. *Mobile Intel Pentium III Processor in BGA2 and Micro-PGA2 Packages Datasheet*, Intel Corporation.
35. S. Moh, C. Yu, B. Lee, H. Y. Youn, D. Han, and D. Lee, "4-ary Tree-Based Barrier Synchronization for 2-D Meshes without Nonmember Involvement," *IEEE Transactions on Computers*, Vol. 50, No. 8, pp. 811-823, August 2001.
36. M. Moudgill, P. Bose and J. Moreno, "Validation of Turandot, a Fast Processor Model for Microarchitecture Exploration," *Proceedings of IEEE Int'l Performance, Computing and Communication Conf.*, pp. 451-457, 1999.
37. "MPEG-2: The Basics of how it Works," Hewlett Packard Lab.
38. W. Namgoong, M. Yu, and T. Meng, "A High-Efficiency Variable-Voltage CMOS Dynamic DC-DC Switching Regulator," *IEEE Int'l Solid-State Circuits Conf.*, pp. 380-381, 1997.
39. O. A. Patino and M. Jimenez, "Instruction Level Power Profile for the PowerPC Microprocessor," *Computing Research Conference 2003*, University of Puerto Rico-Mayagüez, pp. 120-123, April 2003.
40. E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, "Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems," *Technical Report DCS-TR-440*, Dept. Computer Science, Rutgers University, May 2001.
41. Q. Qiu and M. Pedram, "Dynamic Power Management Based on Continuous-Time Markov Decision Processes," *Proceedings of the Design Automation Conference*, pp. 555–561, June 1999.

42. G. Quan and X. Hu, "Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors," *Design Automation Conference*, 2001.
43. L. Shang, L.-S. Peh, and N. K. Jha, "Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks", *Ninth International Symposium on High-Performance Computer Architecture (HPCA'03)*, February 2003.
44. D. Shin and J. Kim, "Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications," *IEEE Design & Test of Computers*, Volume 18, Issue 2, pp. 20-30, March 2001.
45. T. Simunic, L. Benini, P. Glynn, and G. De Micheli, "Dynamic Power Management for Portable Systems," *Proceedings of the International Conference on Mobile Computing and Networking*, pp. 11–19, 2000.
46. D. Son, C. Yu and H. Kim, "Dynamic Voltage Scaling on MPEG Decoding," *International Conference on Parallel and Distributed Systems (ICPADS)*, 2001.
47. M. B. Srivastava, A. P. Chandrakasan, and R.W. Brodersen. "Predictive System Shutdown and Other Architecture Techniques for Energy Efficient Programmable Computation," *IEEE Transaction on VLSI Systems*, Vol. 4, No. 1, pp. 42–55, March 1996.
48. M. Stan and W. Burleson, "Bus-Invert Coding for Low-Power I/O," *IEEE Transaction on VLSI*, Vol. 3, No. 1, pp. 49-58, 1995.
49. C. L. Su, C. Y. Tsui, and A. M. Despain, "Low Power Architecture Design and Compilation Techniques for High-Performance Processors", *COMPCON'94*, 1994.
50. K. Suzuki, et al., "A 300 MIPS/W RISC Core Processor with Variable Supply-Voltage Scheme in Variable Threshold-Voltage CMOS," *IEEE Custom Integrated Circuits Conf.*, pp. 587-590, 1997.
51. V. Tiwari, S. Malik, A. Wolfe, and M. T. Lee "Instruction Level Power Analysis and Optimization of Software," *Journal of VLSI Signal Processing*, Vol. 13, Issue 2-3, August 1996.
52. H. Tomiyama, T. Ishihara, A. Inoue, and H. Yasuura, "Instruction Scheduling for Power Reduction in Processor-Based System Design", *Proceedings of Design Automation and Test in Europe (DATE98)*, pp. 855-860, 1998.

53. H.-S. Wang, X.-P. Zhu, L.-S. Peh and S. Malik, "Orion: A Power-Performance Simulator for Interconnection Networks", *Proceedings of MICRO 35*, November 2002.
54. M. Weiser, B. Welch, A. Demers and S. Shenker, "Scheduling for Reduced CPU Energy," *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 13-23, November 1994.
55. V. Wen, M. Whitney, Y. Patel, J. D. Kubiawicz, "Exploiting Prediction to Reduce Power on Buses," *10th International Symposium on High Performance Computer Architecture (HPCA'04)*, 2004.
56. W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The Design and Use of SimplePower: A Cycle-Accurate Energy Estimation Tool," *Proceedings of the Design Automation Conference*, June 2000.
57. T. T. Ye , G. De Micheli , L. Benini, "Analysis of Power Consumption on Switch Fabrics in Network Routers", *Proceedings of the 39th conference on Design automation*, June 2002.
58. Stemm, M, et al.: "Reducing power consumption of network interfaces in hand-held devices", *Proceedings mobile multimedia computing MoMuc-3, Princeton*, Sept 1996.
59. Lettieri P., Srivastava M.B.: "Advances in wireless terminals", *IEEE Personal Communications*, pp. 6-19, February 1999
60. Havinga P.J.M., Smit G.J.M., Bos M.: "Energy-efficient wireless ATM design", *proceedings wmATM'99*, June 2-4, 1999.
61. Udani S., Smith J.: "Power management in mobile computing", *Tech. report MS-CIS-98-26*, Department of Computer Information Science, University of Pennsylvania, August 1996.
62. Smit G.J.M., Havinga P.J.M., van Opzeeland M., Poortinga R.: "Implementation of a wireless ATM transceiver using reconfigurable logic", *proceedings wmATM'99*, June 2-4, 1999.

63. Balakrishnan H., et al.: “A comparison of mechanisms for improving TCP performance over wireless links”, *Proceedings ACM SIGCOMM’96, Stanford, CA, USA, August 1996.*
64. Ebling, M.R., Mummert, L.B., Steere D.C.: “Overcoming the Network Bottleneck in Mobile Computing”, *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications, Dec. 1994, Santa Cruz, CA.*
65. Kistler J.J.: “Disconnected operation in a distributed file system”, *PhD thesis, Carnegie Mellon University, School of Computer Science, 1993.*
66. Bhoedjang, R.A.F., Rühl T., Bal H.E.: “User-level network interface protocols”, *Computer, November 1998, pp. 53-60.*
67. Wei Ye, John Heidemann, D. E. An energy-efficient mac protocol for wireless sensor networks. *INFOCOM, New York, NY, USA IEEE vol.3 (June 2002), pp.1567–1576.*
68. Gang Lu, Bhaskar Krishnamachari, C. S. R. An adaptive energy-efficient and low-latency Mac for data gathering in wireless sensor networks, *18th International Parallel and Distributed Processing Symposium (IPDPS04) vol.18 (April 2004), pp.224.*
69. Joseph Polastre, Jason Hill, D. C., Versatile low power media access for wireless sensor networks. *Second ACM Conference on Embedded Networked Sensor Systems (SenSys). Baltimore, MD, USA: ACM Press vol. 2nd (November 3-5 2004), pp.95–107.*
70. Marco Avvenuti, Paolo Corsini, P. M., Vecchio, A. Increasing the efficiency of preamble sampling protocols for wireless sensor networks. *Mobile Computing and Wireless Communication International Conference, MCWC. vol. 1st (Sept. 17-20, 2006), pp. 117–122.*
71. M. Buettner, E. Yee, G. V. A., and Han, X-mac: A short preamble mac protocol for duty-cycled wireless sensor networks, *International conference on Embedded networked sensor systems (SenSys) vol. 4th (2006).*
72. El-Hoiydi, A., Decotignie, J.-D, Wisemac: An ultra-low power mac protocol for the multihop wireless sensor networks. *Lecture Notes in Computer Science(LNCS) Vol.3121 (2004), pp.18–31.*

73. *ACPI4Linux* [Інтерет-ресурс]. - Режим доступу: <http://phobos.fs.tum.de/acpi/> вільний.
74. Advanced Configuration and *Power* Interface (*ACPI*) [Інтерет-ресурс]. - Режим доступу: <http://www.acpi.info> вільний.
75. Advanced *Power* Management (*APM*) [Інтерет-ресурс]. - Режим доступу: http://www.microsoft.com/whdc/archive/amp_12.mspx вільний.
76. *Earth Simulator* [Інтерет-ресурс]. - Режим доступу: <http://www.es.jamstec.go.jp/esc/eng/index.html> вільний.
77. *IRIX OS* [Інтерет-ресурс]. - Режим доступу: <http://www.sgi.com/developers/technology/irix/> вільний.
78. *Millywatt* [Інтерет-ресурс]. - Режим доступу: <http://www.cs.duke.edu/ari/millywatt/> вільний.
79. *Motorola Dragonball* [Інтерет-ресурс]. - Режим доступу: <http://www.motorola.com/dragonball/> вільний.
80. *Palm OS Emulator* [Інтерет-ресурс]. - Режим доступу: <http://www.palmos.com/dev/tools/emulator/> вільний.
81. *OnNow* [Інтерет-ресурс]. - Режим доступу: <http://www.microsoft.com/hwdev/onnow/> вільний.
82. *SimOS* [Інтерет-ресурс]. - Режим доступу: <http://simos.stanford.edu/> вільний.
83. *Simulink* [Інтерет-ресурс]. - Режим доступу: <http://www.mathworks.com/products/simulink/> вільний.
84. *SPEC JVM98* [Інтерет-ресурс]. - Режим доступу: <http://www.specbench.org/osg/jvm98/> вільний.
85. The Standard Performance Evaluation Corporation (*SPEC*) [Інтерет-ресурс]. - Режим доступу: <http://www.spec.org/> вільний.
86. A. Klaiber, "The Technology Behind *Crusoe* Processors," *Transmeta* Corporation, [Інтерет-ресурс]. - Режим доступу: http://www.transmeta.com/pdfs/paper_aklaiber_19jan00.pdf вільний.