УДК 004.04
А.О. Кукуруза, Д.П. Павлюк, В.В. Сеник, Б.Ю. Шутко
Тернопільський національний технічний університет імені Івана Пулюя, Україна

МЕТОДИ ОПТИМІЗАЦІЇ ПРОГРАМИ

A.O. Kukuruza, D.P. Pavliuk, V.V. Senyk, B.Y. Shutko

PROGRAM OPTIMIZATION METHODS

Оптимізація програми – це процес модифікації програмної системи для більш ефективної роботи і використання меншої кількості ресурсів деякими з її частин [1]. В цілому, комп'ютерна програма може бути оптимізована таким чином, щоб виконуватися швидше, або працювати з меншою пам'яттю, або іншими ресурсами, або споживати менше енергії.

Оптимізація може відбуватися на декількох рівнях [2]. Зазвичай більш високі рівні впливають більше і їх важче змінити пізніше у проекті. Таким чином, оптимізація, як правило, проводиться через удосконалення на вищому рівні до нижчого. Початкове збільшення ефективності велике і досягається меншими зусиллями, а потім воно зменшується і вимагає більше роботи. Тим не менше, в деяких випадках загальна продуктивність залежить від продуктивності низькорівневої частини програми і невеликі зміни на пізній стадії або початок розгляду з низькорівневих деталей можуть значно вплинути. Зазвичай деяка увага приділяється ефективності цілого проекту і, хоча так не завжди, але вважається, що краще оптимізувати пізно,ніж ніколи [3]. На довгострокових проектах, як правило, є циклиоптимізації, де поліпшення однієї області показує обмеження в інший, і їх, як правило, урізають, коли продуктивність прийнятна або ефективність майже не зросте або буде дороговартісною.

Оскільки продуктивність є частиною специфікації програми, то вона розглядається з самого початку, щоб переконатися, що система здатна забезпечити достатню продуктивність, і на початку прототипи повинні мати приблизно прийнятну продуктивність для того, щоб впевнитись, що остаточна система з оптимізацією досягне прийнятної продуктивності. Це іноді пропускають, вважаючи, що оптимізацію завжди можна зробити пізніше, в результаті в системах прототипу, які занадто повільні, і в кінцевих системах є відмови, тому що вони архітектурно не можу досягнути бажаної продуктивності.

Рівень дизайну вважається найвищим. Його можна оптимізувати для найкращого використання наявних ресурсів, враховуючи цілі, обмеження та очікуване використання / навантаження. Вибір дизайну залежить від цілей, сформованих при проектуванні. Вибір платформи та мови програмування відбуваються на цьому рівні, і їх зміна часто вимагає повного переписування, хоча модульна система може дозволити переписати тільки який-небудь компонент. У розподіленій системі, вибір архітектури (клієнт-сервер, однорангова мережа тощо) відбувається на рівні проектування і може бути важкозамінною, особливо якщо всі компоненти не можуть бути замінені (наприклад, старі клієнти, нові елементи інтерфейсу, стара база даних або й модель вцілому).

Наступним рівнем є рівень алгоритмів та структур даних [4]. Враховуючи загальну конструкцію, продуманий вибір ефективних алгоритмів і структур даних (за асимптотичною складністю) і ефективна реалізація цих алгоритмів і структур даних може значно вплинути на продуктивність. Після оптимізації на рівня дизайну, вибір алгоритмів і структур даних впливає на ефективність найбільше, ніж будь-який

інший аспект програми. Зазвичай структури даних важче змінити, ніж алгоритми.

Загальна методика підвищення продуктивності це уникнення роботи [5]. Ще один важливий метод – кешування, особливо мемоїзації [6], що дозволяє уникнути надлишкових обчислень. Через важливість кешування в системі часто є багаторівневий кеш, що може викликати проблеми з використанням пам'яті, а також питання коректності роботи з старими моделями кешів.

За загальними алгоритмами та їх реалізацією на абстрактну машину слідує вибір рівня конкретного вихідного коду і це може мати істотне значення. Деяка оптимізація в теперішній час може бути виконана шляхом оптимізації компіляторів. Це залежить від вихідної мови, мови цільової машини і компілятора, і може бути важкою як для розуміння і прогнозування, так і зміни з часом; це є ключовим місцем, де розуміння компіляторів і машинного коду може підвищити продуктивність.

Між рівнем вихідного коду і рівнем компіляції, є рівень збірки, який налаштовує продуктивність у вихідному коді і компіляторі відповідно. Директиви і прапори збірки, такі як використання препроцесора, дозволяють відключити непотрібні функції програмного забезпечення, оптимізувати процесори для конкретних моделей, апаратні частини або передбачення розгалуження, наприклад [7].

Використання компілятора з оптимізацією призводить до того, що виконувана програма оптимізована настільки, на скільки компілятор може передбачити.

На найнижчому рівні є написання коду, використовуючи мову асемблер [8], яка призначена для певної апаратної платформи і може виробляти найбільш ефективний і компактний код, якщо програміст користується повним набором машинних інструкцій. Багато операційних систем, що використовуються у вбудованих системах, традиційно написані на асемблері з цієї причини. Коли ефективність і розмір менш важливі, великі частини можуть бути написані на мові високого рівня.

Література.

1. Frost R. Modular and Efficient Top-Down Parsing for Ambiguous Left-Recursive Grammars. // 10th International Workshop on Parsing Technologies / Richard Frost, Hafiz Rahmatullah, Paul Callaghan. – Prague , ACL-SIGPARSE, 2007. – С. 109-120.

2. Golub G. H. Some modified matrix eigenvalue problems / G. H. Golub // SIAM Review, 1973. – vol. 15, no. 2. – С. 318-334.

3. Gu Ming A Divide-and-Conquer Algorithm for the Symmetric Tridiagonal Eigenproblem / Ming Gu, Stanley C. Eisenstat // SIAM. J. Matrix Anal. Appl., 1995. – vol. 16, no. 1. – С.172-191.

4. Guand M. A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem / M. Guand, S. C. Eisenstat // SIAM J. Matrix Anal. Appl., 1995. – vol. 16, no. 1. – С. 172-191.

5. Hennessy John L Computer Organization and Design. The Hardware/Software Interface. 5th Edition / John L. Hennessy, David A. Patterson – Morgan Kaufmann Publishers, 2013. – ISBN 978-0124077263.

6. Hyde Randall The Art of Assembly Language. 2nd Edition / Randall Hyde – No Starch Press, 2010. – ISBN 978-1593272074.

7. Isard Michael Distributed Data-Parallel Programs from Sequential Building Blocks // European Conference on Computer Systems (EuroSys) / [Michael Isard, Mihai Budiu, Yuan Yu та ін.]. – Lisbon, Portugal, 2007. – vol. 41, no. 3. – С. 59-72. – ISBN 978-1-59593-636-3.

8. Knuth D. The Art of Computer Programming, Volume 1: Fundamental Algorithms / Donald Knuth - Addison-Wesley, 2011. – 672 с. – ISBN 978-0201896831.