Анализ и оптимизация на софтуерни приложения

Александър Пенев

Васил Василев

Въведение

Съдържание

Защо производителността е важна?

Дава възможност за:

- По-добра интерактивност (време за отговор)
- Възможност за добавяне на повече функционалност
- По-добра скалируемост
- Използване на по-малко ресурси (енергия, памет, ...)

По-добри времена за отговор

- ABS в автомобилите трябва да действа не по-бавно от хидравличната система
- MPEG декодер – трябва да може да декодира минимум 20 кадъра в секунда
- Търсенето в Google – трябва да има резултати за секунди

Ако времената за отговор се разминават с очакванията софтуерната система е неизползваема

Възможност за добавяне на повече функционалност

Ако наличната работа може да се извърши по-бързо, съкратеното времето за отговор би довело до:

- Възможност за добавяне на повече функционалност
- По-висококачествена обработка
- Обработка на по-големи обеми данни

По-добра скалируемост

Добрите програми имат изключително интензивно натоварване:

- От стотици хиляди до милиони потребители ги използват
- Обработват се огромен брой документи и данни

Тези програми трябва:

- Да могат да издържат на повишаващо се натоварване
- Да се справят лесно с неочаквани проблеми, дължащи се на скалирането

Използване на по-малко ресурси

Повече инструкции за изпълнение води до повече изразходвана електроенергия

- Много от центровете със суперкомпютри са енерго-ограничени.

  През 2005 приблизително 1.2% от електрозахранването в САЩ е предназначено за суперкомпютри

- Обръща се сериозно внимание на мобилните устройства, захранващи се от батерии.

  Батерията на най-новия iPhone издържа по-малко от 1 ден. Това означава, че не може да се добавя нова функционалност, ако не се удължи животът на батерията или приложенията не станат по-ефективни

Скалирането на неефективни системи е много скъпо!

Оптимизация

Например:

- Архитектът, проектиращ многоетажна сграда трябва да избере материалите и пропорциите за различните структурни компоненти на сградата, за да има безопасна сграда, която да е колкото може по-евтина
- Агрономът, ръководещ дадена плантация трябва да планира операциите върху растенията така, че да направи добивът максимален, като задоволява всички нужди на клиентите с минимални средства

Оптимизация

В примерите лесно се забелязва:

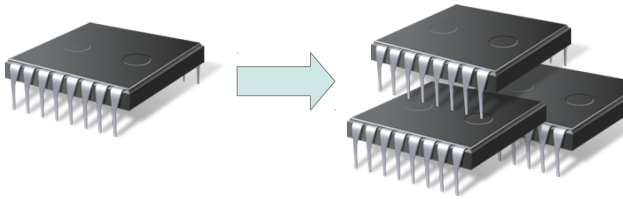
1. Има крайна цел
   За архитекта, крайната цел е да минимизира разходите по построяване на сградата. За агронома – да максимизира добива
2. Има изисквания или ограничения, които трябва да бъдат изпълнени
   Архитектът трябва да изпълни стандартите за безопасност, а агрономът трябва да задоволи нуждите на клиентите, като се съобрази с наличната работна сила и ограничените материали
3. Косвено е идеята, че има възможност за избор и ако е направен успешно ще бъде изпълнена крайната цел и изискванията едновременно

Оптимизация

Неформално оптимизацията включва:

- Избиране на една или повече оптимизационни променливи
- Избиране на целева функция
- Определяне на множество ограничения

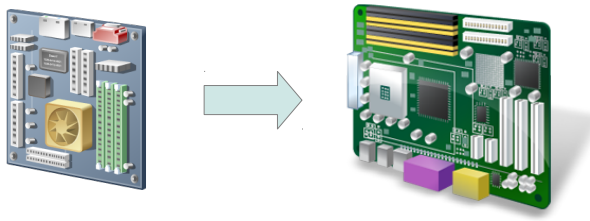
Оптимизиране на софтуер



Извършва се по два начина:

- Хардуерно

  Заменят се хардуерните компоненти на компютърната система с нови, по-производителни



- Софтуерно

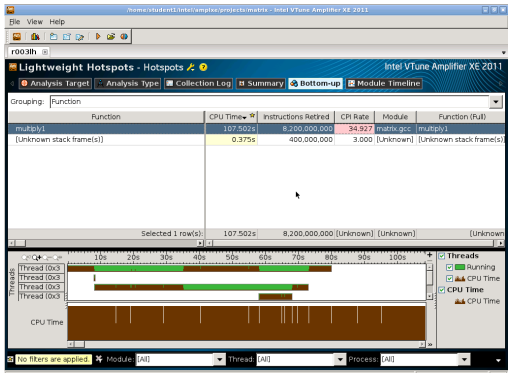
  Създават се по-оптимални (по-ефективни) програми или се оптимизират вече съществуващи

  - Автоматизирано

    Оптимизацията се извършва от софтуерна система, специално разработена за целта

  - Ръчно

    Оптимизацията се извършва от експерт



Оптимизирането на софтуер не е лесно

Стъпки:

- Установяване, че има проблем с производителността
- Идентификация на местата, влошаващи производителността
- Установяване на основните причини, причиняващи проблемите
- Елиминиране на установените проблеми

Установяване, че има проблем

Знаем как да намерим чрез тестване, валидация и верификация

Обаче, колко близо е програмата до най-производителната възможна?

- Трудно е да се каже дори и когато програмата може да се оптимизира много

Как да разберем дали производителността е добра?

- Чрез груби изчисления
- Чрез анализ на производителността
- Чрез тестване на скалируемостта
- Чрез сравнения с подобни програми
- Опит

Идентифициране на местата, влошаващи производителността

Профайлинг (Profiling) на целевата програма

- Установяване къде се губи най-много време
  - Очаква ли се? Или има проблем?
- Преглед на характеристиките на машината
  - Нормално ли е поведението на инструкциите, кеша, паметта?

Тестване за скалируемост

- Натоварване на програмата до краен предел
- Установяване кое може да се скалира и кое не

Ако профайлинга променя прекалено много производителността резултатите не са надеждни!

Установяване на причините

Изследване на алгоритъма

- Прекалено неефективен ли е алгоритъма
- Може ли някое от изчисленията да бъде елиминирано от критичния път
  - Използвайки препроцесорна обработка, кеш...

Изследване на структурите от данни и тяхното оформление (layout)

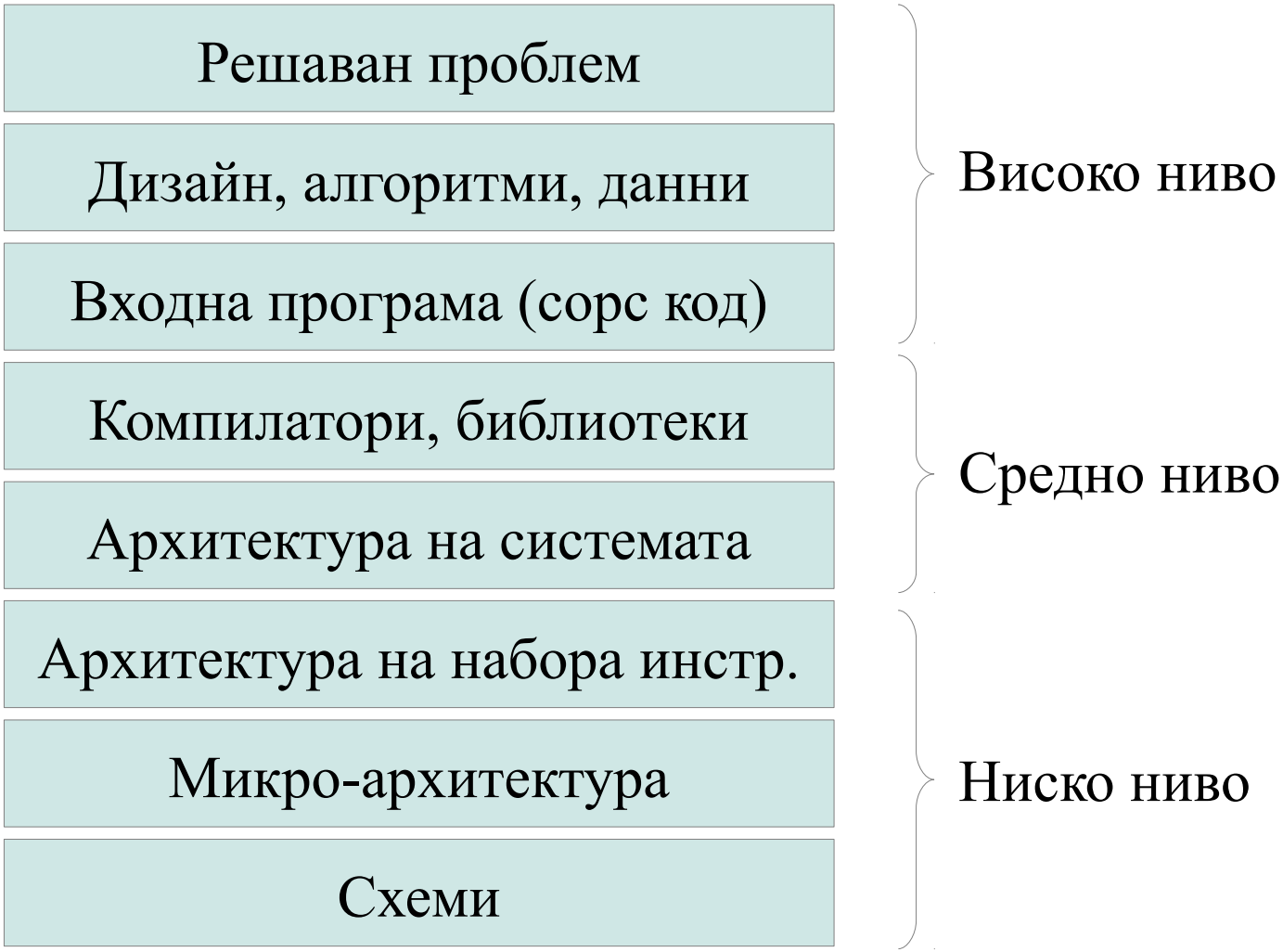
- Дали оформлението влияе на паметта?

Изследване на структурата на програмата

- Дали структурата на програмата не води до генерация на лоши инструкции и забавяне на конвейера (stalls)

Много от нещата се решават с опити и грешки

Обща схема


Решаван проблем
Дизайн, алгоритми, данни
Входна програма (сорс код)
Компилатори, библиотеки
Архитектура на системата
Архитектура на набора инстр.
Микро-архитектура
Схеми
Високо ниво
Средно ниво
Ниско ниво

Елиминиране на проблемите

Дизайн

- Много по-добре е дизайнът да се съобразява с производителността

Неспазване на идеите в дизайна

- Понякога трябва да се нарушат добрите практики, залегнали в дизайна

Придържане към основните принципи

- Простота, модулност, преносимост и т.н
- Не може да е за сметка на коректността

Елиминиране на проблемите

Има нужда да се разбират всичките слоеве и тяхното значение и специфики

- Всички софтуерни слоеве
- Компилатора
- Процесора
- Системата

Трябва да се спазват основните принципи:

- Не може да е за сметка на коректността