Увод

Рачунарски систем

Рачунарски систем

Рачунарски систем

- Хардвер - софтвер
- Физичка архитектура - програмска подршка
- Изучавају се често одвојено, али у рачунарској техници су тесно повезани и док се бави једним увек се мора мислити на друго.
- Пошто развој физичке архитектуре често претходи развоју програмске подршке, развој програмске подршке више зависи од физичке архитектуре него обрнуто.

Рачунарски систем

- Кад расте величина рачунара, расте и цена
- Кад расте величина рачунара, расте и потрошња

- Рачунарски систем треба ради **што боље**, да буде што јефтинији и да што мање троши.

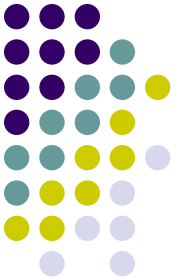
Рачунарски систем

- Кад расте величина рачунара, расте и цена
- Кад расте величина рачунара, расте и потрошња

- Рачунарски систем треба ради **довољно добро**, да буде што јефтинији и да што мање троши.

- **Шта** треба да ради довољно добро?

Рачунарски системи



- Системи за рад у реалном времену
- Уграђени системи
- Системи са ограниченим ресурсима -> Процесори са ограниченим ресурсима
- **Наменски рачунарски системи** -> Наменски процесор

Системи у реалном времену

- Системи (за рад) у реалном времену.

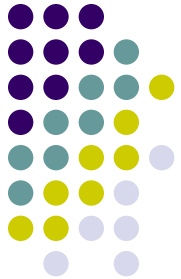
- Приликом пројектовања готово је немогуће направити оштру разлику између „шта“ и „како“.

Уграђени системи

- Нема спреге човека и машине (или је сведена на минимум)
  - Дакле, нема: тастатуре, екрана итд.

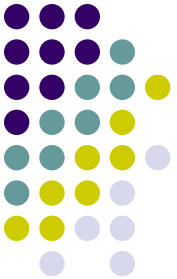
- 99% процесора је у уграђеним системима
- Животни циклус система: развој, производња, одржавање.
- Код уграђених система цена развоја је обично испод 5% укупне цене целог циклуса.
- Економски аспекти имају далеко већи значај

Системи са ограниченим ресурсима



- Ресурси:
  - Меморија, регистри
  - **Брзина**
  - Енергија
  - ...
- Ограниченост је релативна
- али радни такт уграђених система је обично до неколико стотина мега-херца

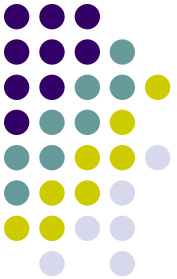
Наменски системи



- Системи посебне намене

- Наменски процесори
  - Сваки процесор сабира бројеве... ?

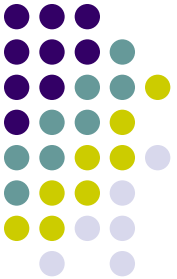
Груписање процесора по намени



- Процесори опште намене

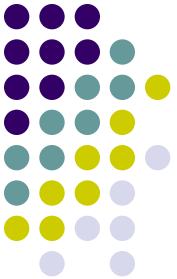
- Наменски процесори (Процесори посебне намене)
  - ДСП-ови
  - Микроконтролери
  - Графички процесори...

Процесори опште намене



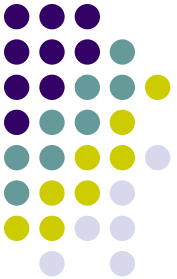
- Најбоље извршавају све могуће врсте програма

Процесори опште намене



- Најбоље извршавају све могуће врсте програма
- Подједнако добро извршавају све могуће врсте програма

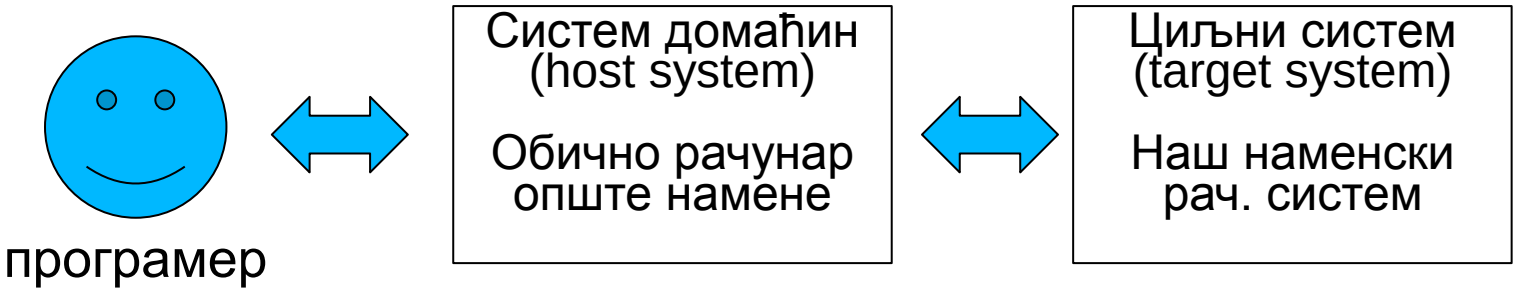
Процесори опште намене



- Најбоље извршавају све могуће врсте програма
- Подједнако добро извршавају све могуће врсте програма
- Неке врсте програма извршавају боље, неке лошије, али је разлика обична мања од једног реда величине (10x)

Чиме се програмирају наменски рачунарски системи

- Рад са HPC најчешће није директан.



- Део системског софтвера се извршава на домаћину.
- Домаћин је систем са више ресурса и већом брзином, а пре свега комотнији за рад.
- Многи елементи системског софтвера ни не би могли бити извршавани на циљном систему.

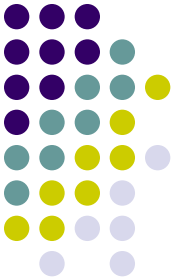
Чиме се програмирају наменски рачунарски системи

- Алати:
  - Асемблер
  - Компајлер (за виши програмски језик)
  - Дебагер (компонента за контролисано извршавање програма)
  - Симулатор (омогућава бољу контролу и бољи увид у извршавање кода)
  - Интегрисано развојно окружење
- Оперативни систем и библиотеке

Када се програмирају наменски рачунарски системи

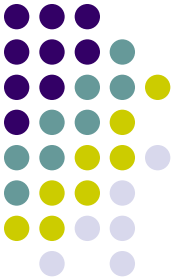
- Пре него што је физичка архитектура готова
  - Да би се убрзао развој система
  - Део кода је већ раније развијен
- Када је физичка архитектура фиксирана, али још увек није произведена
  - Развојне плоче са прототипом
  - Симулатори
- Када је физичка архитектура готова, али системски софтвер још није (алати)
  - Развој системског софтвера обично креће пре него што је физичка архитектура готова
- Када је физичка архитектура спремна и зрела
  - Код неких система ово је већ крај животног циклуса софтвера
  - Код неких дугих - нови почетак

Како се програмирају наменски рачунарски системи



- Зависи од много фактора:
  - Фазе у развоју - спремност физичке архитектуре и алата
  - Уложеног развојног напора - у развој системског софтвера, алата пре свега
  - Обима конкретног програма - за мали програм може и бинарно да се програмира
  - Захтевности конкретног програма - на нижем нивоу апстракције се може више исцедити из архитектуре

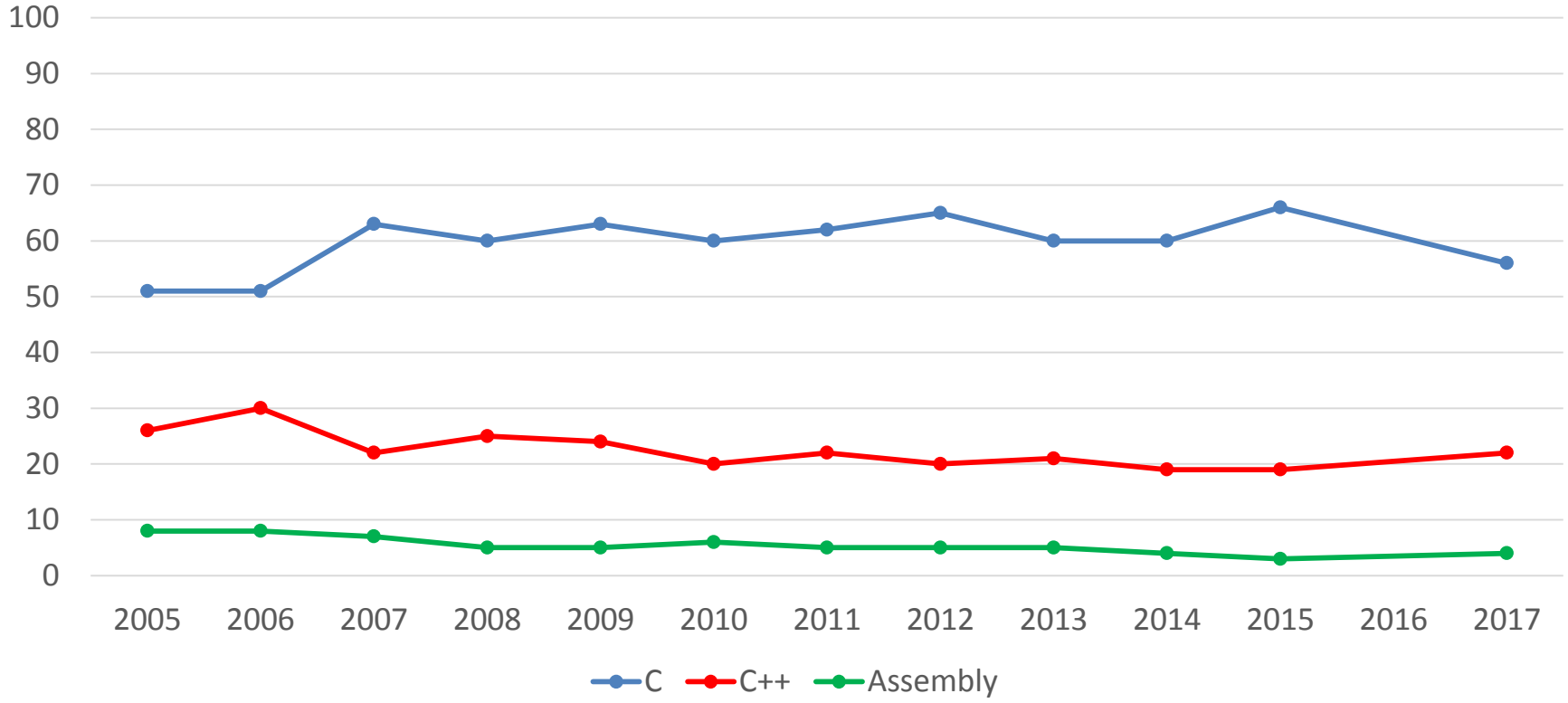
Како се програмирају наменски рачунарски системи



- Асемблер
  - Ближи архитектури
  - Преводилац лакши за прављење
  - Компликованији за успешно програмирање
- Виши програмски језик
  - Даљи од архитектуре
  - Преводилац компликованији
  - Програмирање лакше

Прогамски језик Це

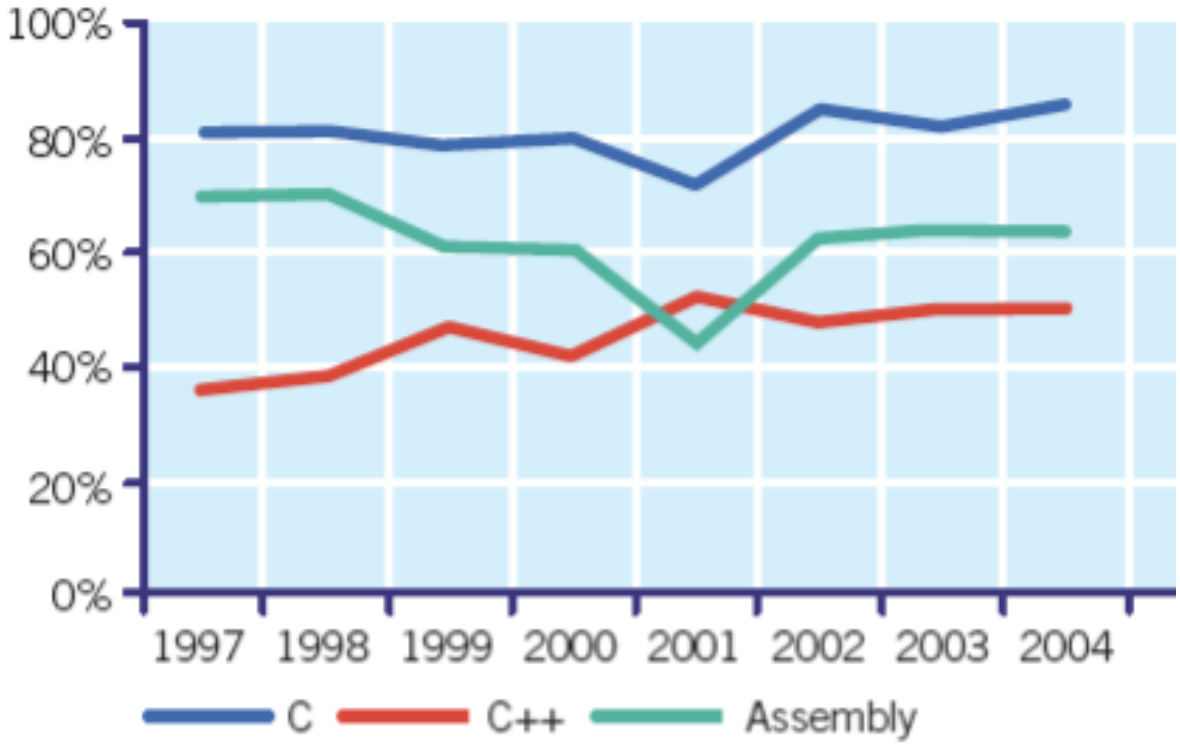
- Виши програмски језик који се најчешће користи за програмирање HPC је Це.



Проценат пројеката у којима се већински користи дати програмски језик

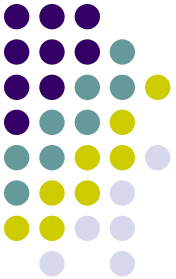
Прогамски језик Це

- Виши програмски језик који се најчешће користи за програмирање HPC је Це.



Проценат пројеката у којима се користи дати програмски језик

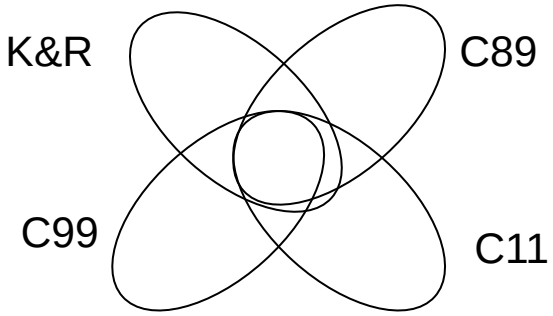
Прогамски језик Це

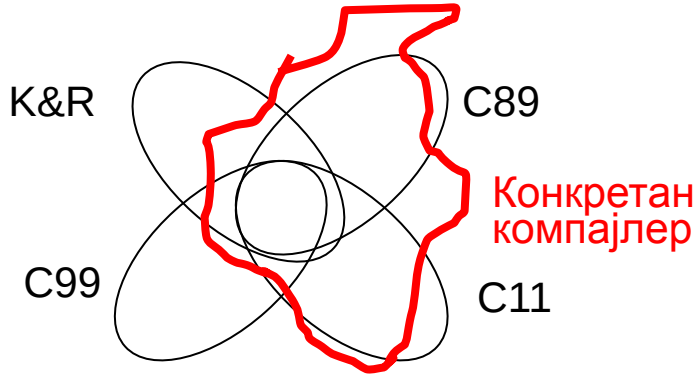


- Зашто програмски језик Це?
  - Није превише висок
    - Омогућује боље цеђење перформанси
    - Релативно добро пресликавање његових операција на операције физичке архитектуре
  - Историјски разлози
    - Нагомилан већ постојећи код
    - Велики број његових познаваоца
    - Конструкција компајлера релативно једноставнија

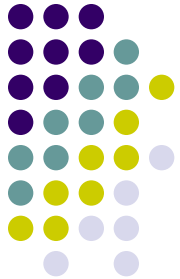
Прогамски језик Це

- Зашто **НЕ** програмски језик Це?
  - Није превише висок
    - Виши ново апстракције би олакшавао програмирање
    - Релативно **лоше** пресликавање његових операција на операције физичке архитектуре
  - Историјски разлози
    - Ваљда је до сада могло бити смишљено нешто паметније
    - Велики број његових „познаваоца”
    - Који стандард тачно подржава конкретан систем?





Циљ предавања и вежби

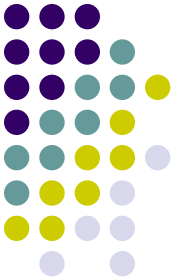


Оспособити студенте да могу:

- Писати нови Це код за HPC
  - Научити делове језика који нису у фокусу када се програмирају системи опште намене и када се ад хок програмира, али су важни при програмирању HPC
  - Учврстити добру праксу, зарад писања лепог, јасног и употребљивог кода
  - Препознати сиве зоне језика да их не бисте користили
- Разумети постојећи Це код за HPC
  - Препознати сиве зоне језика да бисте их разумели у коду са којим се будете сусретали

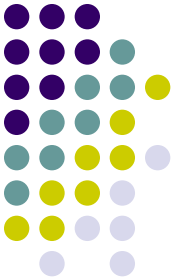
Организација наставе

- Предавања
  - [9. 10. 2017. – 18. 10. 2017.] од 9 до 12.
- Вежбе
  - [9. 10. 2017. – 18. 10. 2017.]
  - 42. група 13-16 у Л1, 41. група 13-16 у Л2 и 43. група 16-19 у Л1
- Израда пројекта
  - [19. 10. 2017. – 25. 10. 2017.]
  - консултације у терминима вежби
  - одбрана 26. и 27. 10. 2017. од 13 до 19
- Тест
  - 3. 11. 2017. у 10ч.
- Формирање оцене:
  - 5% присуство, 45% пројекат, 50% испит



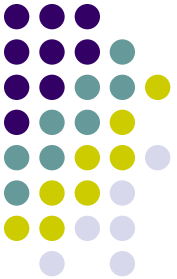
Дебаговање

Дебаговање?



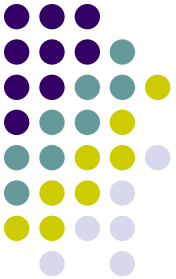
- Буба (енгл. Bug /баг/) - Проблем у програму
- Отклањање буба, проблема
- У ужем смислу то подразумева: контролисано извршавање програма и увид у стање програма зарад отклањања проблема.
- У ширем смислу то су сви могући (и немогући) поступци за откривање и отклањање проблема.

Тестирање (испитивање)



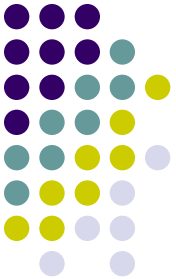
- Програм најчешће ради савршено све док га не покренемо.
- Провера исправности: верификација и валидација
- Верификација проверава да ли програм ради то што ми мислимо да треба да ради.
- Валидација проверава да ли програм ради то што нама треба. (То што ми мислимо да програм треба да ради не мора бити то што нама треба.)
- Тестирање не мора бити формализовано! Али је то више него пожељно код озбиљних подухвата.

Отклањање проблема

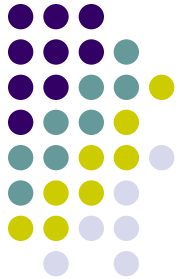


- Програм најчешће не ради добро све док га не издебагујемо.
- Испитни случај: улаз (побуда) - очекиван излаз (реакција)
- Зашто добијамо неочекиван излаз?
- Потребан нам је бољи увид у извршавање програма.
- Уобичајено доступне ствари:
  - Контролисано извршавање:
    - Корак по корак и тачке прекида
  - Увид у стање програма
    - Посматрање променљивих и меморије
    - Додавање контролних исписа у код

Отклањање проблема



- Неуобичајено доступне ствари:
  - Контролисано извршавање:
    - Корак у назад
    - Условне тачке прекида
    - Кретање по току података, уместо по току извршавања
  - Увид у стање програма
    - Срачунавање израза у тренутном контексту
- Али неки пут немамо ништа од поменутог!
  - Тада се сналазимо како знамо и умемо.



Неке поделе багова

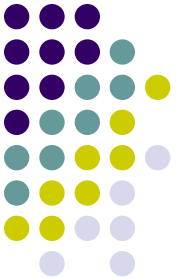
- Ухватљиви (доследни, поновљиви) багови

  За одређену побуду увек добијамо исту нежељену реакцију
  - Лако поновљиви багови
    - Побуду која узрокује нежељену реакцију је лако направити у контролисаним условима
  - Тешко поновљиви багови
    - Побуду која узрокује нежељену реакцију је тешко направити у контролисаним условима
- Неухватљиви (недоследни, стохастички) багови

  За одређену побуду нежељену рекцију добијамо само понекад

  Узрок је постојање нежељеног утицаја на систем ван чинилаца побуде
  Могући узроци:
  - Читање из неиницијализоване меморије
  - Ослањање на адресе динамички заузете меморије
  - Упадање у стања недефинисана програмским језиком

Дигресија: недефинисана стања и понашања

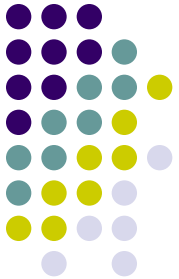


- То значи да понашање може бити различито приликом извршавања истог кода на истом систему.

Неки важнији примери недефинисаних понашања у програмском језику Це:

- Прекорачење код интеџерске аритметике
  - Укључујући померање у десно, тзв. шифтовање
- Битске операције над интеџерима
  - Бинарна репрезентација интеџера не мора бити комплемент двојке!
- Важи само за означене интеџере!
  - Неозначени се дефинишу тотално другачије.

Неке поделе багова



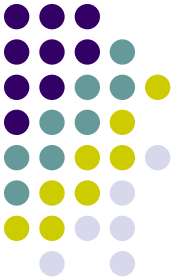
- Прости багови

  Манифестација проблема је тесно везана за место где постоји грешка у коду.

- Сложени багови

  Манифестација проблема и место грешке у коду су врло посредно повезане.

Како успешно дебаговати



- **Разумети систем**
- **Репродуковати баг**
- **Не претпостављати - погледати**
- **Подели па владај**
- Постепено мењати
- Водити белешке
- Проверити кабел (да није до физичке архитектуре?)
- Применити нов поглед на ствари
- Ништа се не поправља само од себе

Најважнији савет!

Покушати решити проблем