Бележки към избора и реализацията на проектите (r2.0).

(1) Избор на проект.

Стандартният избор на проект, става чрез съобщение/писмо до rmi@yaht.net.

Ако все пак имате собствено виждане за това какво Ви се пише и сте убедени, че можете да изкарате ускорение от него (т.е. да го накарате да използва повече от един процесор/нишка), моля запознайте се с изискванията в следващите точки и цитираните примерни документации, след което ме уведомете на id@yaht.net.

(2) Какво е ускорение и как всъщност го мерим?

При разработването на програми използващи паралелни процеси (или нишки) се нуждаем от мерило, което да ни показва с колко сме подобрили работата на програмата. За целта сравняваме серийната (не - паралелна) версия на програмата със паралелната версия на програмата, като критерия за сравнение е времето им за изпълнение. Измервайки времето за изпълнение на програмите, определяме ускорението на паралелната, спрямо серийната версия на програмата. С други думи, колко пъти по-бързо се изпълнява нашата паралелна версия, спрямо серийната такава.

(2.1) Закон на Amdahl (Амдал)

Теоретичен модел за очакваното ускорение на дадена паралелна програма, спрямо серийната версия на програмата ни дава закона на Амдал.

Законът има вида: $$Speedup = \frac{1}{(1-p) + \frac{p}{n}}$$

Какво всъщност ни казва този закон?

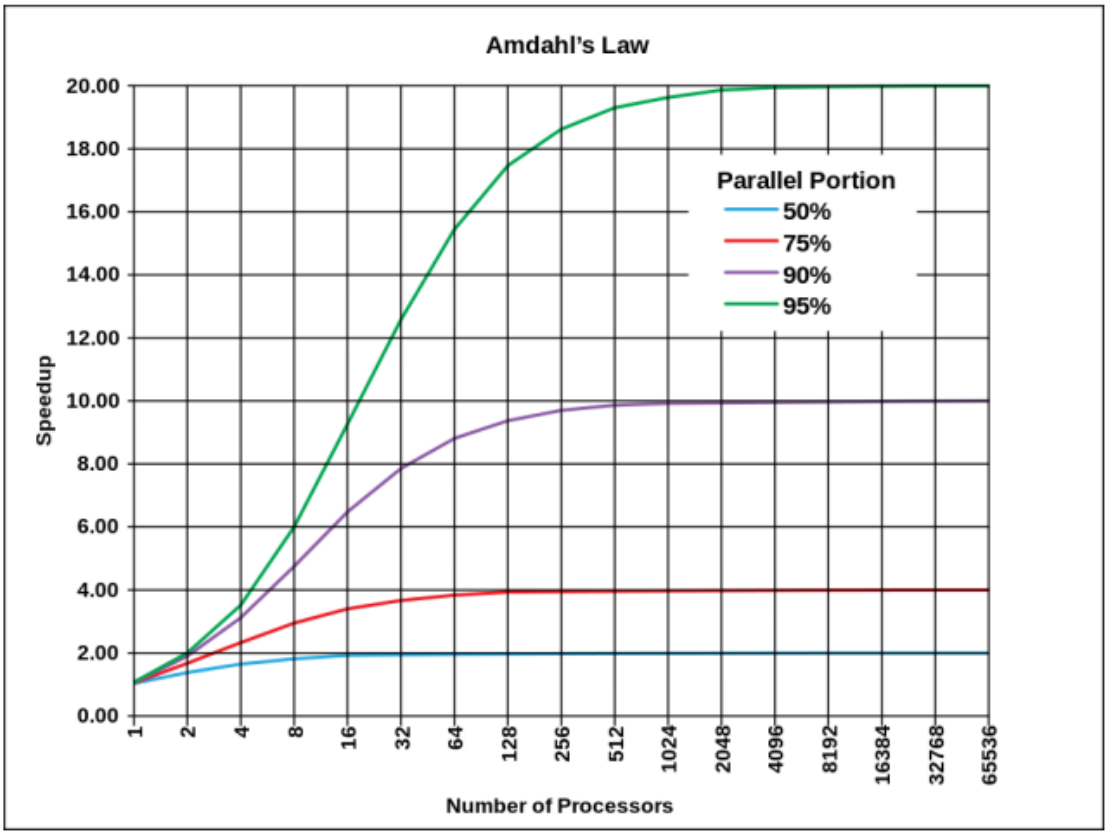
Във формулата с **p** означаваме отношението на времето прекарано в „раз-паралеленият код“ към „цялото време за изпълнение“ на програмата (цялото време за изпълнение е 100% или 1). С **n** означаваме броя процесори, които кода може да използва.

Цялото време през което е работила нашата паралелна програма е сума от следните две времена: времето прекарано в не-паралелна работа (**1-p**) и времето прекарано от всеки процесор вършейки паралелна работа (**p/n**).

Законът на Амдал е просто отношение, което ни показва как паралелизма (паралелното изпълнение на програмата) е ограничен от необходимостта от серийно изпълнение (последователно, не-паралелно).

Когато програмата ни няма паралелен код (**p = 0**) - ускорението е 1. Тоест това не е паралелна програма. Когато програмата няма необходимост от синхронизация или друг сериен код (**p = 1**), тогава ускорението е **n** (броят на процесорите с които разполагаме). Колкото повече синхронизация е необходима, толкова по-малко облаги получаваме от паралелното изпълнение на програмата. Казано по друг начин нашата програма ще работи по добре, на повече процесори, ако моделира независими (или слабо зависими) една от друга задачи.

Най-добре очакванията давани от закона на Амдал, могат да бъдат видени в графичен вид:



Друго важно наблюдение, която можем да направим, разчитайки на графичният вид, е следното – при условие че в програмите, които пишем винаги има серийна част (тоест не са и не могат да бъдат 100% паралелни) и при условие че увеличаваме борят процесори на разположение на нашата програма, то съществува точка (т. нар. точка на насищане), преминаването на която не гарантира по нататъшно ускорение на програмата.

Тоест с колкото и ресурс да разполагаме, не можем да ускорим програмата, повече от някаква фиксирана горна граница. Това на свой ред ни дава възможност да се замислим дали сме реализирали най-добрата паралелна версия на нашият, обикновено сериен алгоритъм ;).

(2.2) Реално измерими параметри.

Тъй като параметрите изисквани от закона на Амдал са много трудно измерими (да не кажа невъзможно), се нуждаем от друг начин да мерим ускорението постигано от нашите програми.

Нека:

T1 – времето за изпълнение на серийната програма (или програмата използваща една нишка/един компютър).

Tp – времето за изпълнение на паралелната програма, използваща p – процесора/нишки/компютри.

Тогава:

Sp = T1/Tp – е ускорението, което нашата програма има при използването на p процесора/нишки/компютри.

Ep = Sp/p – е ефективността (ефикасността) на нашата програма, при използването на p процесора/нишки/компютри.

Как можем да измерим **T1**, ..., **Tp**?

Примерът който ще дам, разчита на езика **Java**.

Разглеждаме класа **Calendar (java.util.Calendar)**. Методът който ни интересува е:

long getTimeInMillis()

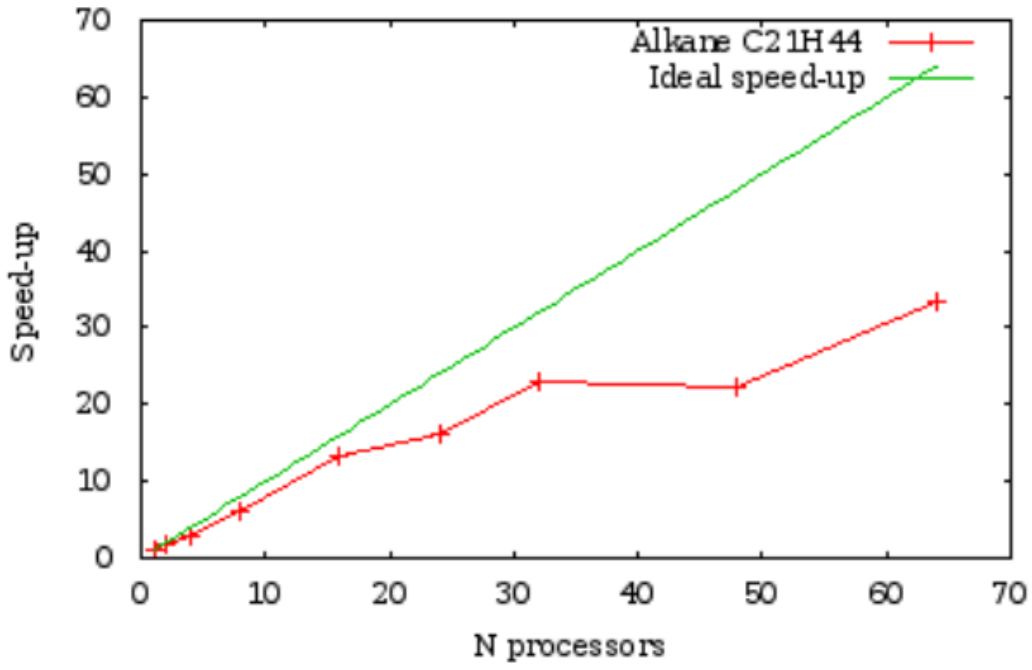
Методът връща времето на системата (в милисекунди), от началото на UNIX Epoch (1970-01-01 00:00:00). Разчитайки на този метод (**Calendar.getInstance().getTimeInMillis()**), вземаме две времена:

t1 – в началото на изпълнението на нашата програма.
t2 – в края на изпълнението на нашата програма (след като и последната нишка е приключила изпълнението си)

Разликата **t2 – t1** е времето на изпълнение на нашата програма (в милисекунди).

Събирайки времената от тестовете на на програмата (**за p = 1 ... 32**), можем лесно да оформим табличен вид на резултатите, по който в последствие да построим диаграма (графика) на ускорението и ефективността на нашата програма.
Сравнително лесно това става, като разчитаме на Microsoft Excel или Open Office/Libre Office.

Разчитайки на събраните тестови данни от програмите и някои от изброените офис пакети, очакваме следните диаграми (графики) за ускорението на програмите ни:



БЕЛЕЖКА: Диаграмите (графиките), които ще представите да бъдат във вид **Line Chart (Area Chart) за Sp, Tp и Ep**. Нека всеки параметър да бъде изобразен на отделна диаграма. Ако алгоритъмът, който реализирате търпи развитие, то всяка следваща версия на ускоренията му, може също да бъде изобразена в диаграма. По този начин ще покажете дори визуални подобренията, които сте направили (например на една и съща диаграма **Sp1** и **Sp2** за двете версии на алгоритъма).

(3) Изисквания към документацията.

Примерни документации на вече предадени проекти, можете да намерите тук: http://rmi.yaht.net/docs/example.projects

r2.0 на изискванията – условното разделение на проектите е вече в 5 категории: **bad, not.so.bad, good, excellent, own.choise**.

bad – документации на проекти, които аз не бих си позволил да предам /т.е. сигурно е че съм имал забележки към тях и съм връщал колегите за extra fixes/;

not.so.bad - документации на проекти, които отново не бих си позволил да предам :(, като разликата с предишната точка е че не съм връщал колегите за extra fixes;

good – добри документации, минали без забележки;

excellent – мисля че се подразбира от името :). Обикновено хората, които познават добре кода и езика на който пишат, представят и документации съответстващи на качеството на програмите им ;);

own.choise – документации на проекти, които не са измежду раздаваните по случаен начин, но въпреки това отговарят на изискванията от точка (2) за ускорение, което освен това е показано по подходящ начин.

Отделно от това, моите насоки за документацията на Вашите проекти са:

(o) Размер на документацията – 5,7,9 страници, без да цитирате код, по възможност. Не че ще Ви се разсърдя ако е повече, но този обем е предостатъчен да обясните алгоритъма си и да представите исканите диаграми ;).

(o) Алгоритъмът който реализирате - task decomposition || data decomposition || both.

(o) Архитектура на приложението - диаграма на нишките, например master/slave relation(s), кой кого пуска, чака, уведомява и т.н. Последното като картинка и/или обяснение.

Шаблони (design patterns) за използване на нишки - ако са използвани някакви е добре да ги знаете какво правят и дори опишете :), тъй като често пъти използването им води до някой много неприятни ефекти в програмите Ви (което лъсва веднага в резултатите от работата на програмата и диаграмите към тях);

(o) Speed up информация (т.е. Tp/Sp/Ep) на база резултатите от тестовете. Тоест за една и съща големина на задачата (харесвате си някаква), времето за изпълнение на програмата, съответно пусната със **1, 2, 3, ... 8, ..., 12, ..., 16 , ..., max 32** нишки. Тестовите резултати подредете в табличен вид и задължително в графичен, генериран с някой от изброените по горе офис пакети. Моля не забравяйте за Tp/Sp/Ep.

(o) Ако добавите code snippets от Вашето решение /но не целият source code, моля ;)/ със сигурност ще преминете споменатия обем от 5,7,9 страници. В това число включвам началната страница и условието на задачата, разбира се :).

(o) Хората реализирали chat clients/server би следвало да се съсредоточат във описанието си върху client-server архитектурата на приложението и съответно описание на собствената им реализация. От chat clients/server няма как да „изсмучем“ ускорение, освен ако не слезем на много, много ниско ниво.

(o) Проф. Георгиев също е дал насоки за оформяне на документациите към проектите. Моля запознайте се и с тях. Както ще се убедите документациите от excellent секцията покриват тези критерии.

(o) Моля качете крайните версии на документацията и програмите си на тестовият сървър (на поне един от 3-те);

(4) Изисквания към реализацията.

Единственото изискване към реализациите Ви е да покажат ускорение, на някоя от тестовите машини или на машина с близки до техните параметри.

В този смисъл очаквам да сте тествали с поне **8, ..., 32** нишки и то на машина която го позволява /долната граница е 8, а не горната :)/.

Използвам „позволява“ имайки в предвид, че няма смисъл да претоварвате дву-ядрен (четири-ядрен) процесор с **8, 16, 32** .. т.н. нишки, тъй като просто не разполагате с хардуер който да удовлетвори едновременната им работа. Тоест не очаквайте голям ефект в/у тестовите си резултати в този случай :).

Единственото ограничение към езика на който ще реализирате проектите си е да "върви" в GNU/Linux среда. Трите машини, до които бота раздава достъп са със CentOS GNU/Linux. Ако все пак пишете на език, който го няма в стандартната дистрибуция на CentOS, ще го доставим с помощта на Docker.

Текстовете на задачите са написани с идеята да се ползва Java, но това е само препоръчително. Ако сте избрали език, който има поддръжка в GNU Linux, но го няма инсталиран на тестовата системата, моля пишете ми за да го инсталирам.

Краен срок за предаване и защита са датите на устните изпити за всяка от специалностите /вкл. последния възможен момент, от последната възможна дата :)/.

Все пак дати ще обявя допълнително на http://rmi.yaht.net.

За потвърждаване на резултатите от документациите към проектите ще използваме тестовите машини, до които имаме достъп – тоест добре е програмите да могат да се компилират и там.