Проект

по Системи за Паралелна Обработка

Паралелен Алгоритъм за Изчисляване на E

Изготвил:
Йосиф Цветков

Ръководител:
ас. Христо Христов

Проверил: .....
(ас. Христо Христов)

1. Постановка на задачата

Едно важно за математиката число е Неперовото число (Ойлеровото число), тоест числото e. Използвайки сходящи редове, можем да сметнем стойността на e с произволно висока точност. Един от сравнително бързо сходящите към e редове е:

$$e = \sum_{k=0}^{\infty} \left(\frac{2k+1}{(2k)!}\right)$$

Вашата задача е да напишете програма за изчисление на числото e използвайки цитирания ред, която използва паралелни процеси (нишки) и осигурява пресмятането на e със зададена от потребителя точност. Изискванията към програмата са следните:

(o) Команден параметър задава точността на пресмятанията. По Ваше желание, точността се изразява или в брой цифри след десетичната запетая или в брой членове на реда. Командният параметър задаващ точността има вида „**-p 10000**";
(o) Друг команден параметър задава максималния брой нишки (задачи) на които разделяме работата по пресмятането на e – например "**–t 1**" или "**–tasks 3**";
(o) Програмата извежда подходящи съобщения на различните етапи от работата си, както и времето отделено за изчисление и резултата от изчислението (стойността на ***e***);
(o) Записва резултата от работа си (стойността на ***e***) във изходен файл, зададен с подходящ параметър, например "**-o result.txt**". Ако този параметър е изпуснат, се избира име по подразбиране;
(o) Да се осигури възможност за „**quiet**" режим на работа на програмата, при който се извежда само времето отделено за изчисление на ***e***, отново чрез подходящо избран друг команден параметър – например "**-q**";

ЗАБЕЛЕЖКА:

(o) При желание за направата на подходящ графичен потребителски интерфейс (**GUI**) с помощта на класовете от пакета **javax.swing** задачата може да се изпълни от **двама души**; Разработването на графичен интерфейс не отменя изискването Вашата програма да поддържа изредените командни параметри. В този случай към функцията на параметъра параметъра „**-q**" се добавя изискването **да не пуска** графичният интерфейс. Причината за това е, че Вашата програма трябва да позволява отдалечено тестване, а то ще се извършва в **terminal**.

2. *Описание на реализирания алгоритъм*

За паралелното решаване на задачата е избран подходът на многонишковото програмиране. Очевидно решение би било да се пресмята всеки терм, използвайки се формулата и акомулиране на резултата. Поради неефиктивноста на този подход, в настоящетата документация са разгледани няколко подхода за решаването на този проблем. За целта е използван програмния език Go, версия 1.4.2

2.1.1 Първа идея

Първото наблюдение което ще направим и многократно ще ползваме(в следващите версии на програмата) е това че не е нужно да използваме „чистия“ вид на формулата, спомената по-рано, за изчисляването на всеки терм от реда. Вместо това, ако положим a_k да значи к-тия терм от реда, то бихме да забележим че:

$$\frac{a_{(k+1)}}{a_k}=\frac{(2(k+1)+1)*2k!}{(2k+1)*(2(k+1))!}=\frac{2k+3}{(2k+1)^2(2k+2)} \rightarrow a_{(k+1)}=\frac{a_k(2k+3)}{(2k+1)^2(2k+2)}$$

Това ни дава зависимост между съседните термове, която е много по-проста от изчислителна гледна точка спрямо оригиналната, а също така и алгоритмична т.е. асимптотичната сложност е доста по-ниска. Но тъй като изчисляването на кой да е терм зависи от предишния, ние все още се намираме в доста „последователна“ среда. Затова бихме могли да обобщим току-що изведената формула по следния начин:

$$\frac{a_{(k+t)}}{a_k}=\frac{(2(k+t)+1)(2k)!}{(2(k+t))!(2k+1)}=\frac{(2k+2t+1)((2k)!)}{(2k+2t)!(2k+1)}=\frac{2(k+t)+1}{(2k+2t)\dots(2k+2)(2k+1)^2}=\frac{2(k+t)+1}{(2k+1)\prod_{i=1}^{2t}(2k+i)}$$

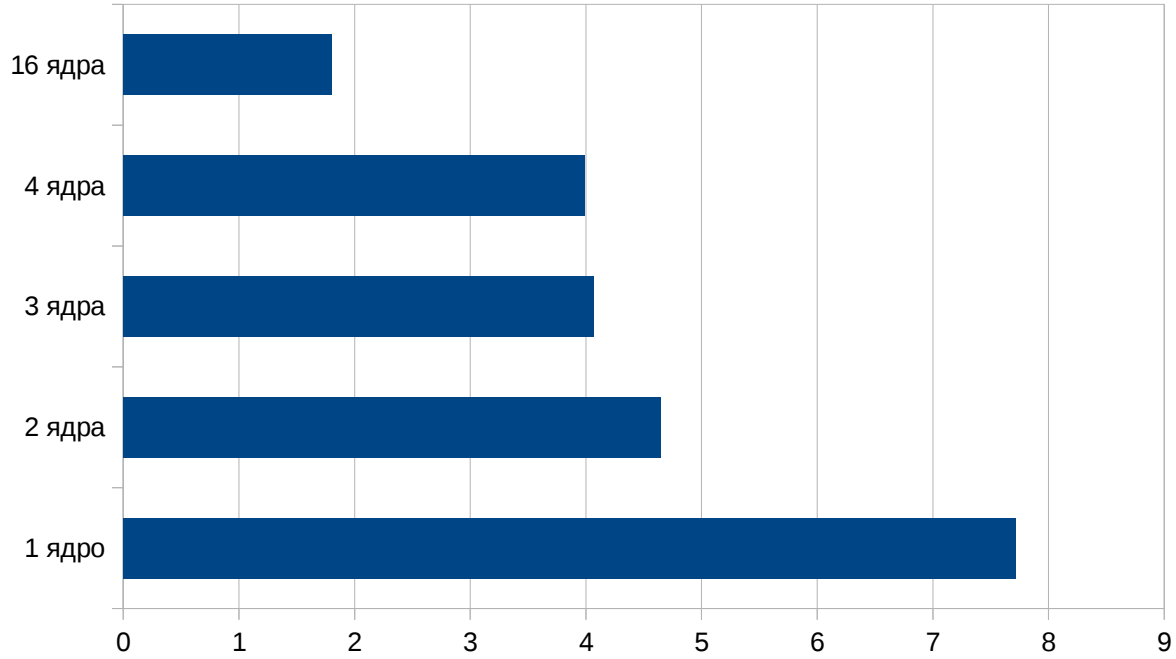
където „t“ е параметър, изразяващ разлика между разтоянието на два терма в оригиналния ред. За по-напред, ще се обръщам към първата от тези формула като формулата за съседни термове, а към втората, като формулата за термове на разтояние.

С помоща на тази формули се очератава ясен план за паралелен алгоритъм, а именно:
1) решаваме колко процеса, нишки, или като цяло, колко изчислителни процеса ще ползваме, примерно 't'
2)пресмятаме първите 't' терма от реда изполвайки формулата за съседни термове
3)пускаме 't' на брой изчислителни процеса, всеки с точно един от вече изчислените термове и ичисляваме всеки t-ти член използвайки формулата за термове на разтояние като междувременно акомулираме резултат от вече

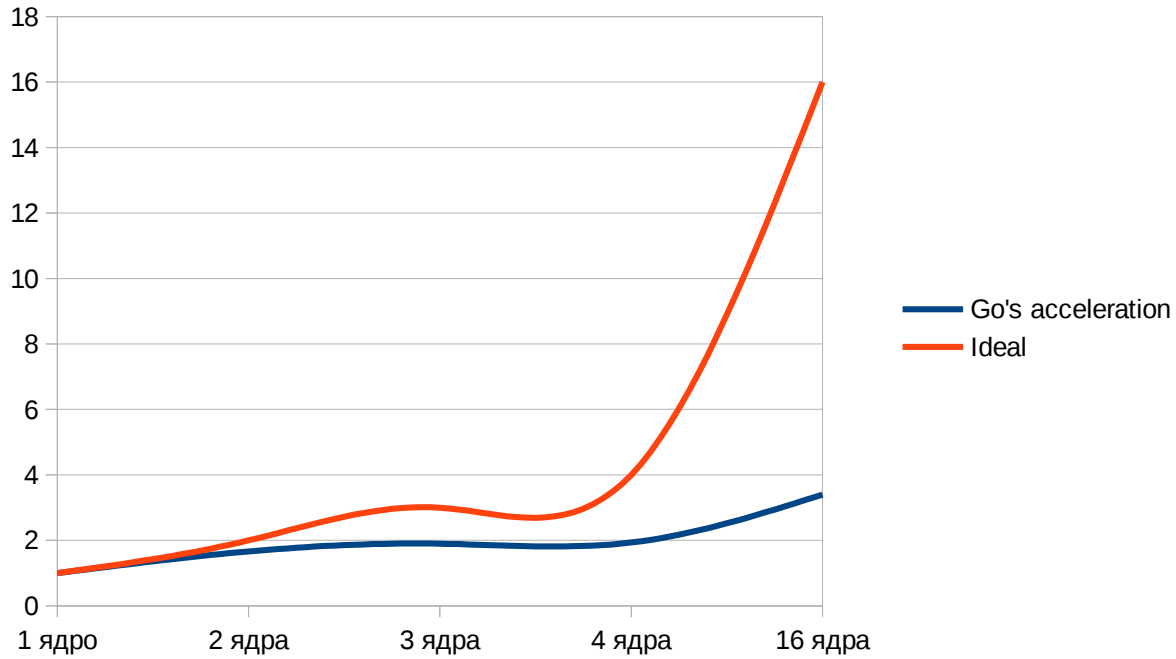
изчислените термове, получавайки 't' на брой частни суми(по един за всеки изчислителен процес), които след като се сумират, дават като резултат желаната константа.

2.1.2 Резултати

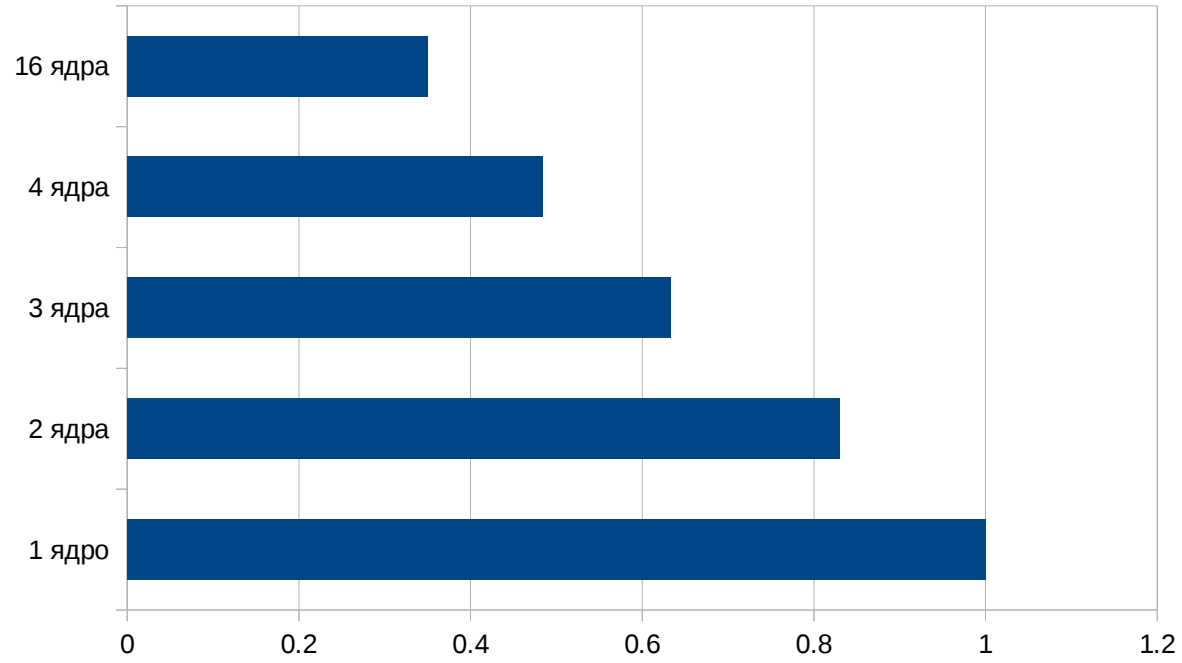
Следните графики показват какви скорости и ускорения може да се очаква с гореспоменатия алгоритъм.



Фиг. 1 *Показва средното време за изчисление на даден вход*

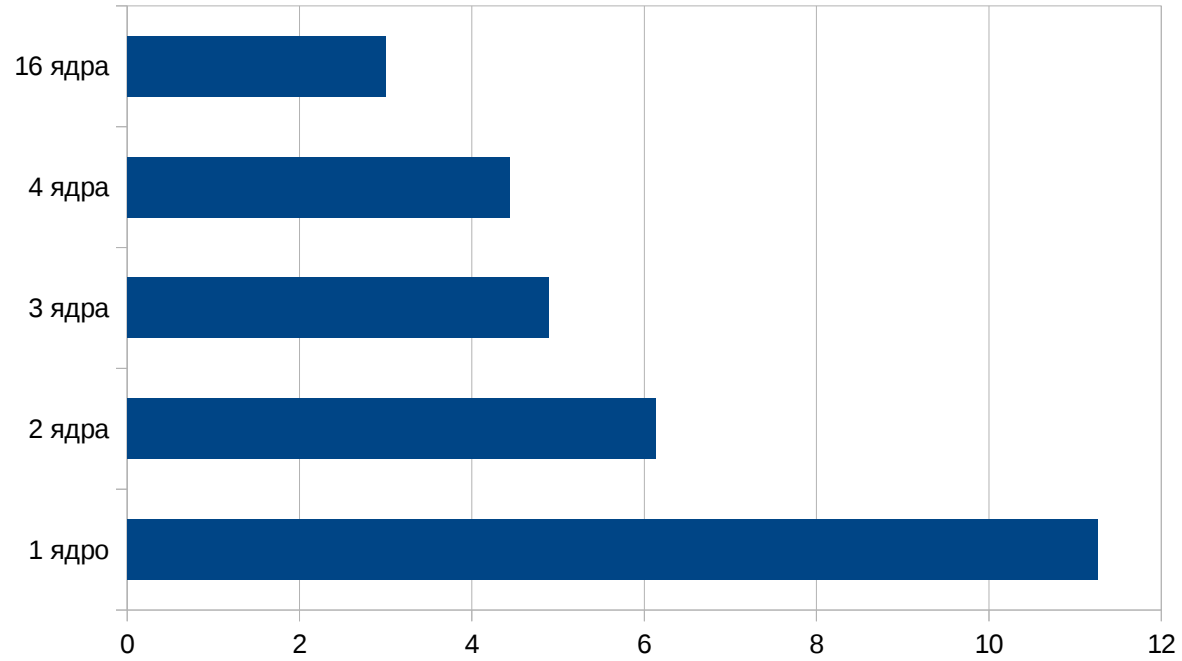


Фиг. 2 *Показва ускорението на програмата*

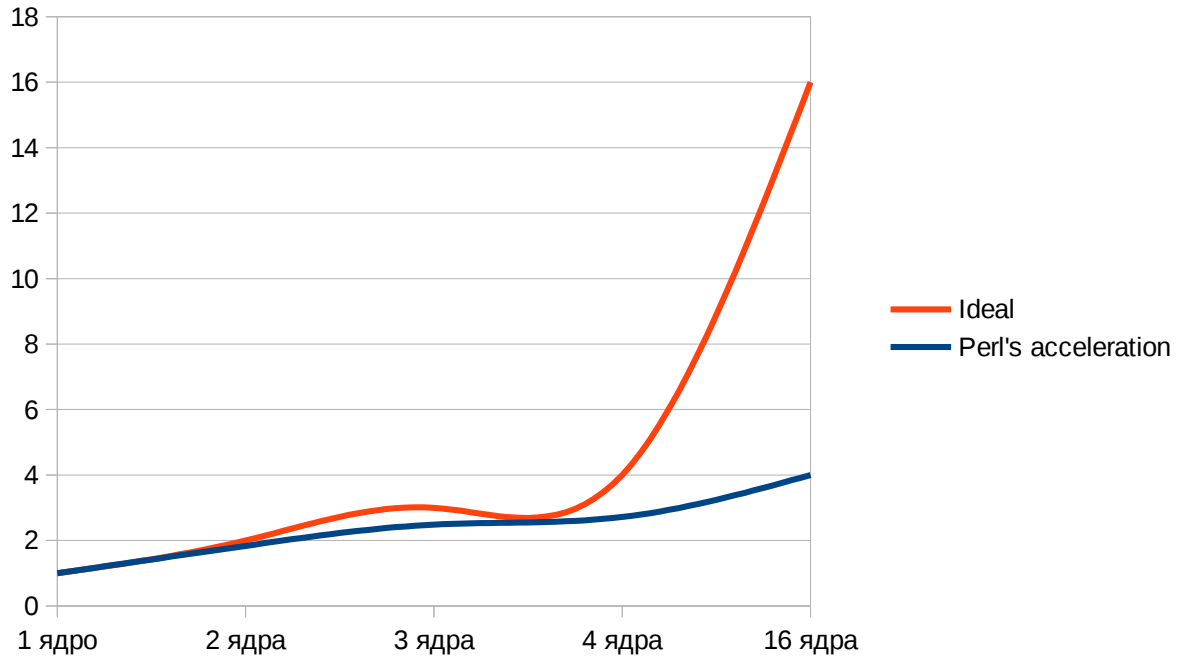


Фиг. 3 Показва ефективността на програмата

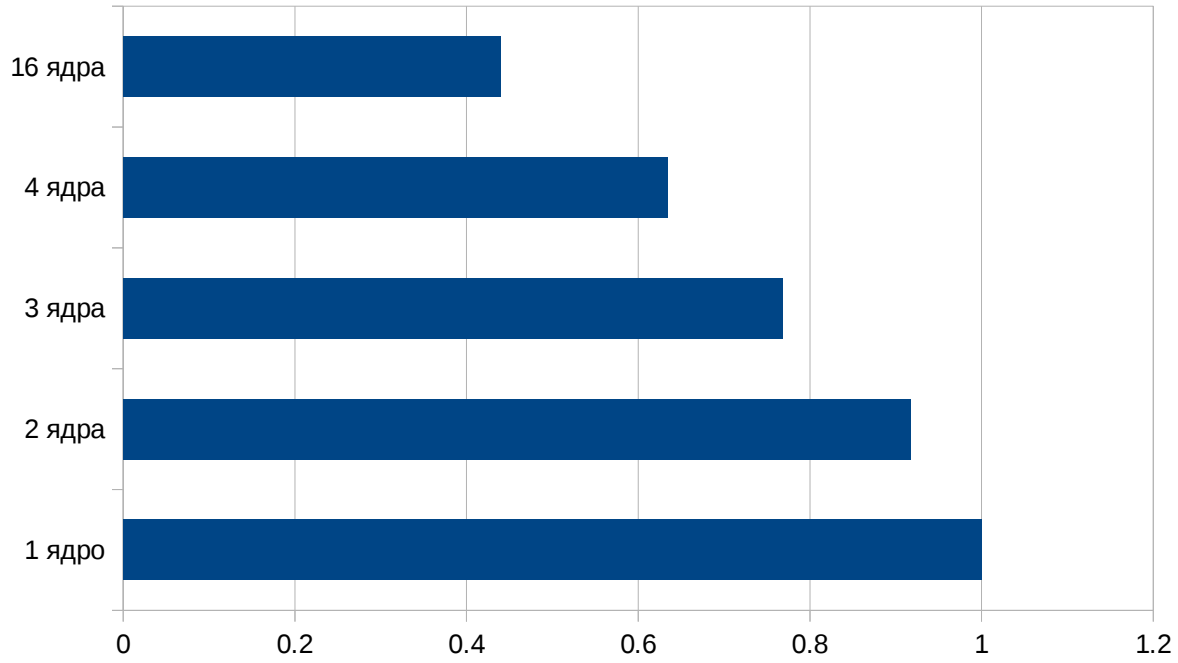
Забелязва се, че ускорението е доста ниско. Също така е важно да се каже, че получените резултати се основават на сравнително малък вход.
Бързодействието на програмата е под средното за задача от този тип.
Използвайки същата идея и подход но друг език, и по-точно Perl, получаваме следните резултати:



Фиг. 4 *Показва средното време за изчисление използвайки Perl*

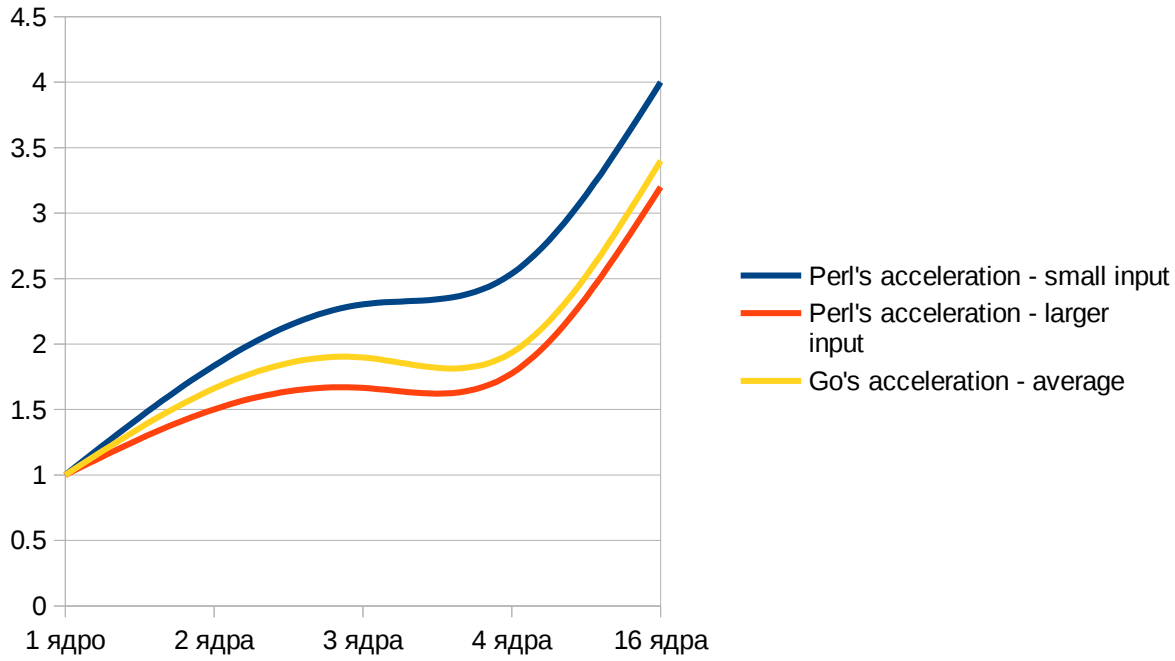


Фиг. 5 Показва ускорението

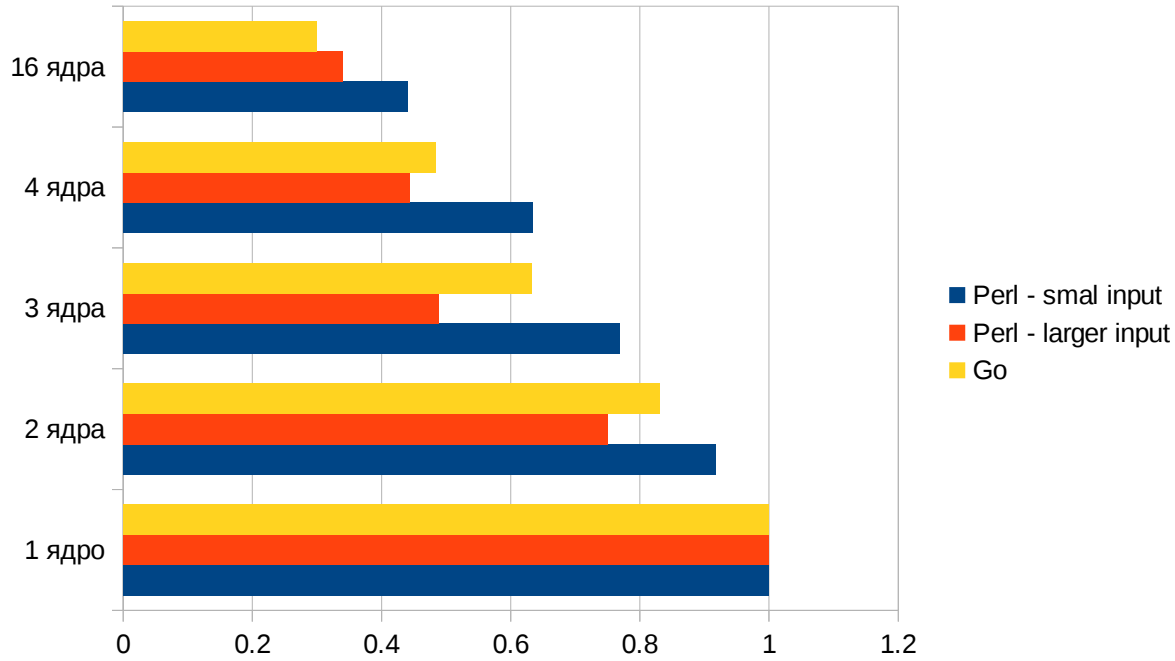


Фиг. 6 Показва ефективността

Въпреки, че получените резултати индикират за наличие на подобрено ускорение, при внимателен анализ и тест бихме видяли че при по-голям вход, данните от Perl-ската програма започват да клонят към тези от тази написана на Go. Това може да се забележи от следните графики:



Фиг. 7 Обща графика на ускоренията



Фиг. 8 Обща графика на ефективността

Същите резултати, с точност до константи, се наблюдават и при реализации използвайки C++ и JavaScript. По това може да се съди за това че ускорението не се влиае от програмния език, а само от това до колко е адекватен алгоритъма. Поради това, от тук нататък ще бъдат показвани резултатите получени от програми написани на Go, с уговорлата че получените резултати ще бъдат представителни за всички реализации.

2.1.3 Анализ

Въпреки, че се забелязва ускорение, то е, както биха се изразили някои хора, нищожно. Вглеждайки се по-внимателно във формулата за извеждане на терм от друг такъв на произволно разтояние, ще забележим проблема, а именно:

$$\prod_{i=2}^{2t}(2k+i)$$

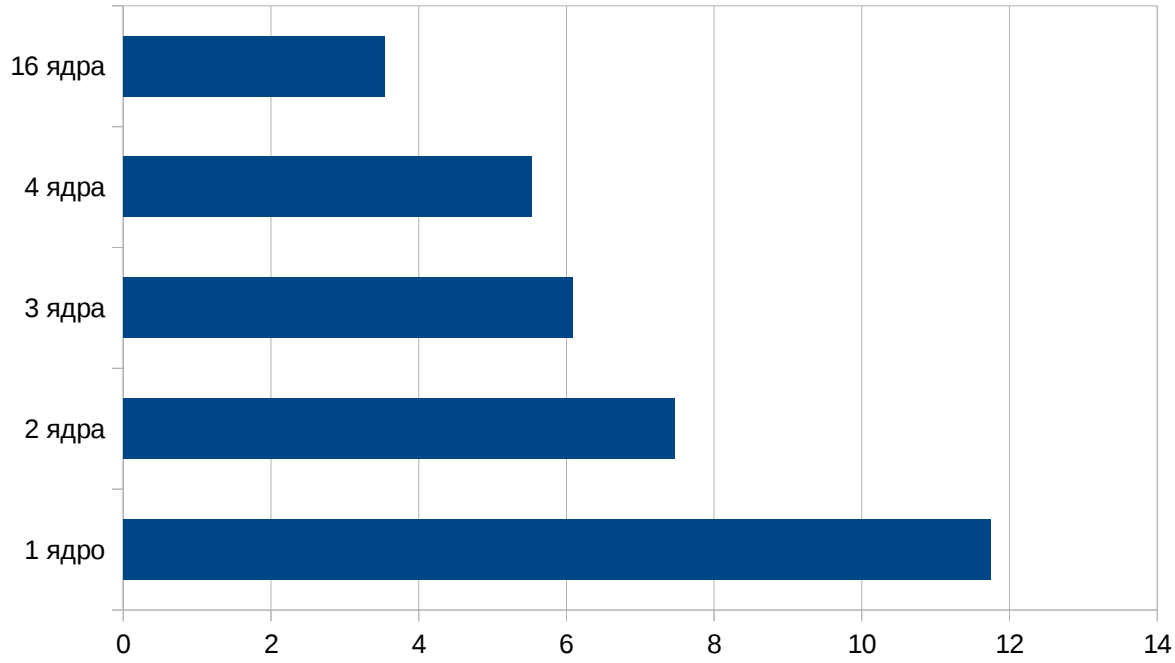
На пръв поглед изглждаща невинно, то при по-обстоен анализ става явно, че по-голям брой изичислителни процеси, натовараването на отделните процеси разте. И то нетривиално. Затова оставяме тази версия и започваме да мислим по друга.

2.2.1 Втори опит

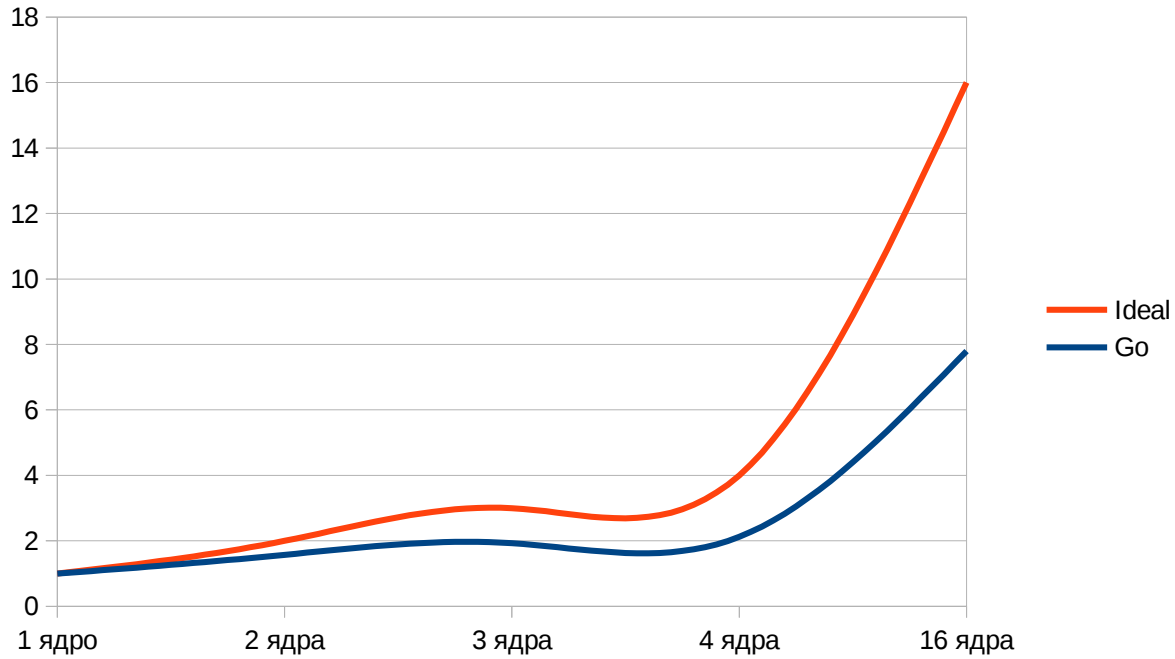
При втория ни опит, все още се опитваме да изчислим константата „***e*“** използвайки алгоритъм от типа „Разделяй и владей“, но този път вместо да пресмятаме отделни термове през интервали, ще пресмятаме цели блокове от термове. За целта отново изчисляваме първите няколко терма(по-точно толкова на брой, колкото изчислителни процеса ще стартираме), и след това тези термове се подават на функции които изчисляват фиксиран брой термове, изпозвайки формулата за съседни термове. Отново всяка функция акомулира сума от термовете които е сметнала, но този път, броя на тези суми е повече от една. За целта се ползва глобална споделена памет където би могло да се записва информация. Различните езици разполагат с различни структури за тази цел затова няма да специфицирам какво да се ползва, а само ще спомена, че реализираната от мен програма на Go ползва така наречените канали. С цел бързодействие и ускорение, след завършването на всеки изчислителен процес, освен частична сума се запаметява и последно изчисления терм от функцията. Целта на това е този терм да се подаде на следващ изчислителен процес който с минимална работа да го преизползва и да продължи работата си.

2.2.2 Резултати

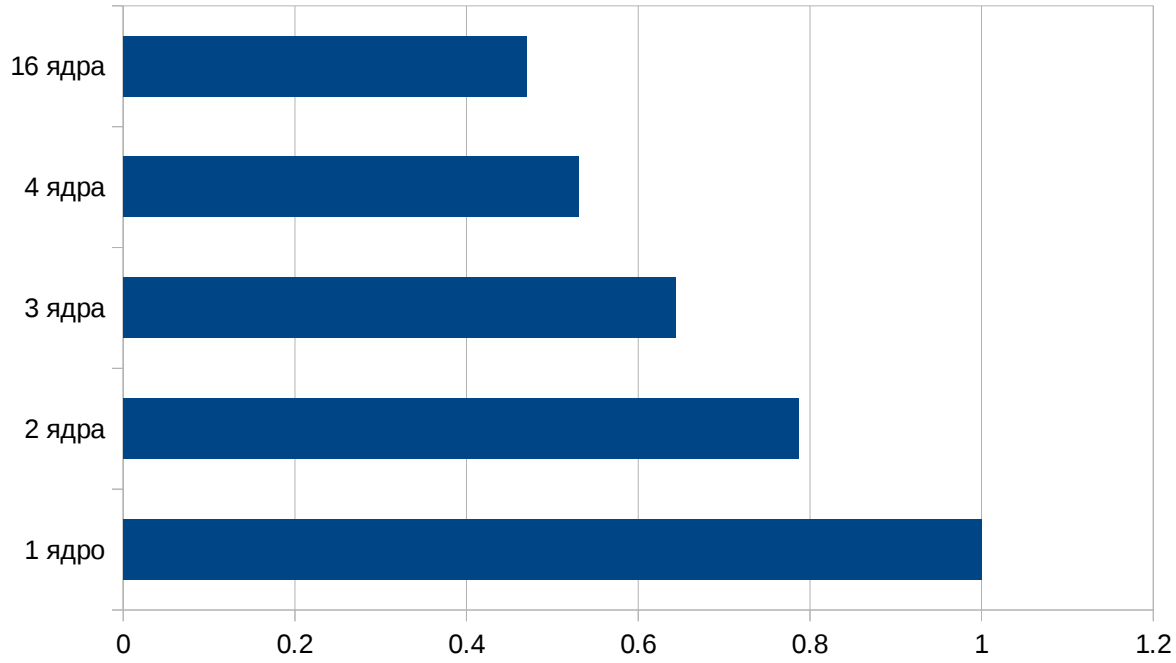
Следните графики показват какви скорости и ускорения може да се очаква с гореспоменатия алгоритъм.



Фиг. 9 Показва ефективността

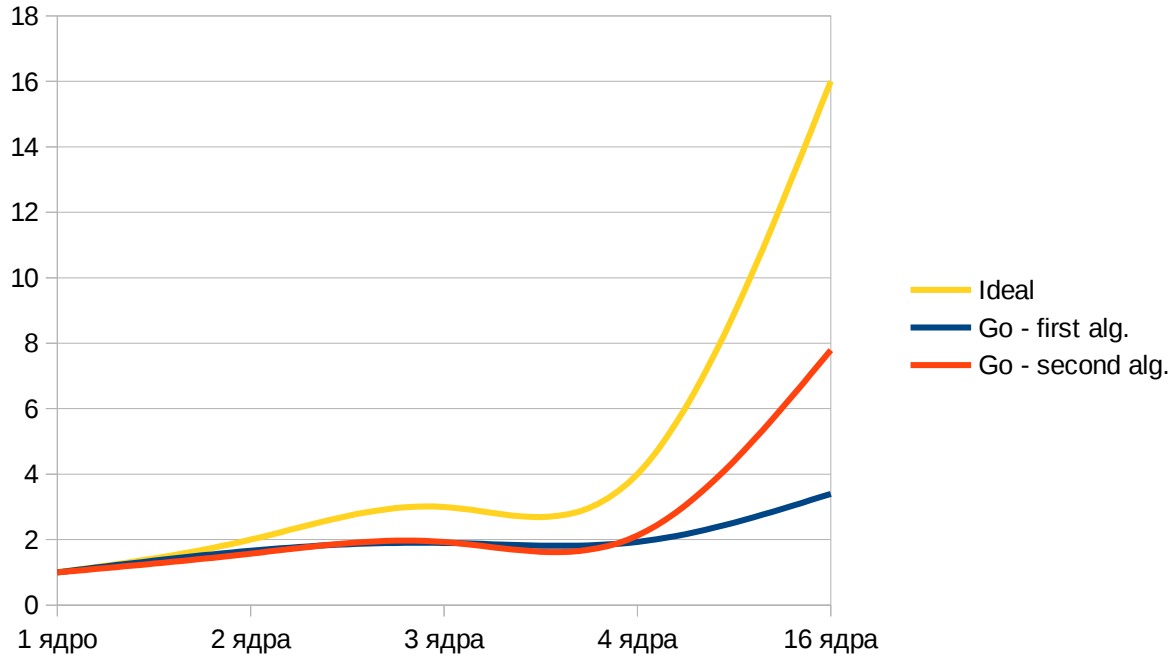


Фиг. 10 Показва ускорението



Фиг. 11 Показва ефективността

Бързодействието на програмата значително се увеличава, признак за което е голямината на входа който може да се обработи за същото време спрямо предишната версия, но ускорението все още не е такова каквото се очаква.



Фиг. 12 Сравнение на ускорението на двата алгоритъма

2.2.3 Анализ

Както бе споменато, ускорението все още не е достатъчно. Причина за това е преизползването на термове което бе споменато. Въпреки добрата и идея и чудесни намерение, реализацията основаваща се на тези идеи не би била

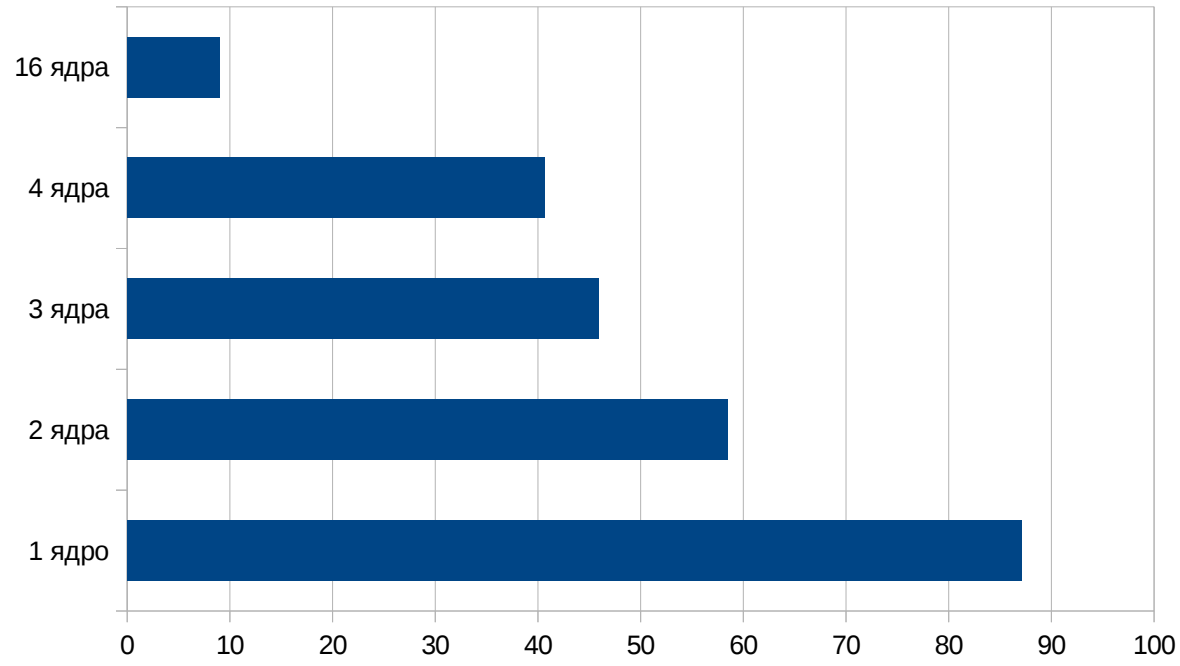
оптимална(което може да се съди по-графиките), тъй като вече изчислен терм би оптимизарл действието на един процес, и само до някъде на останалите, но не достатъчно. При достатъчно голямо интервали(в случая интервали от порядъка на 100 терма) показват, че не е достатъчно.

2.3.1 Трета и последна версия

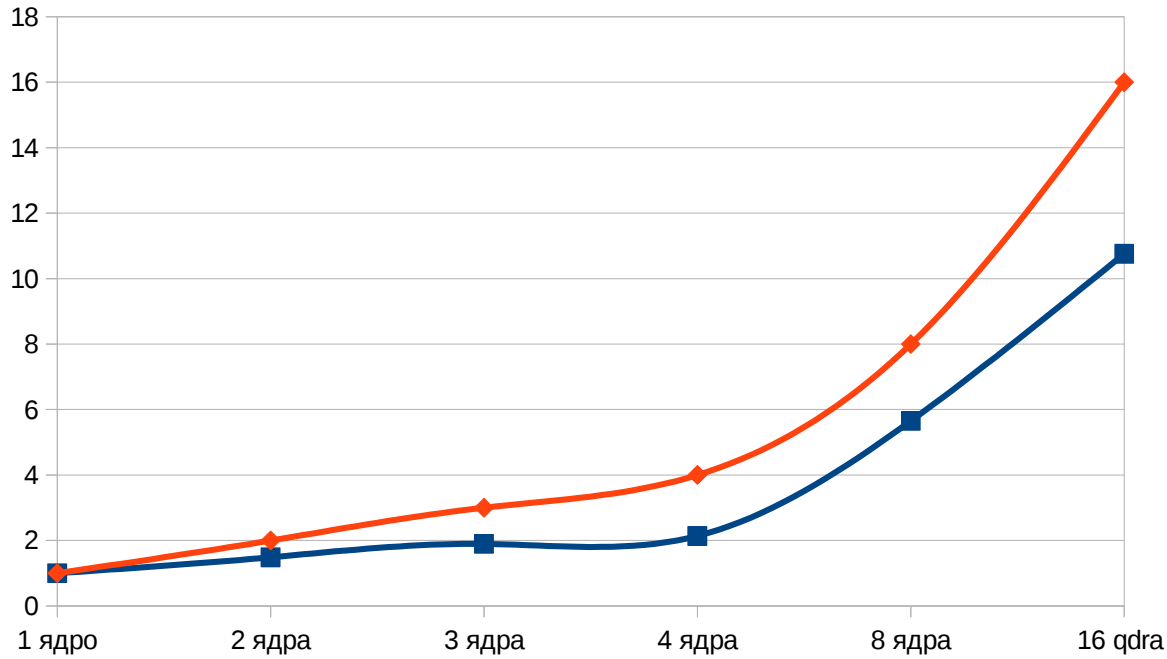
Финалната реализация представлява модификация, или оптимизирана версия на втората програма. Изменението което се внася е да се въведе една глобална променлива, индикираща послено сметнатия терм от кой да е изчислителен процес. Като такъв, изчисленията които извършва кой да е процес се минимизира и натоварването се разпределя по-добре спрямо овеличения брой изчислителни процеси.

2.3.2 Резултати

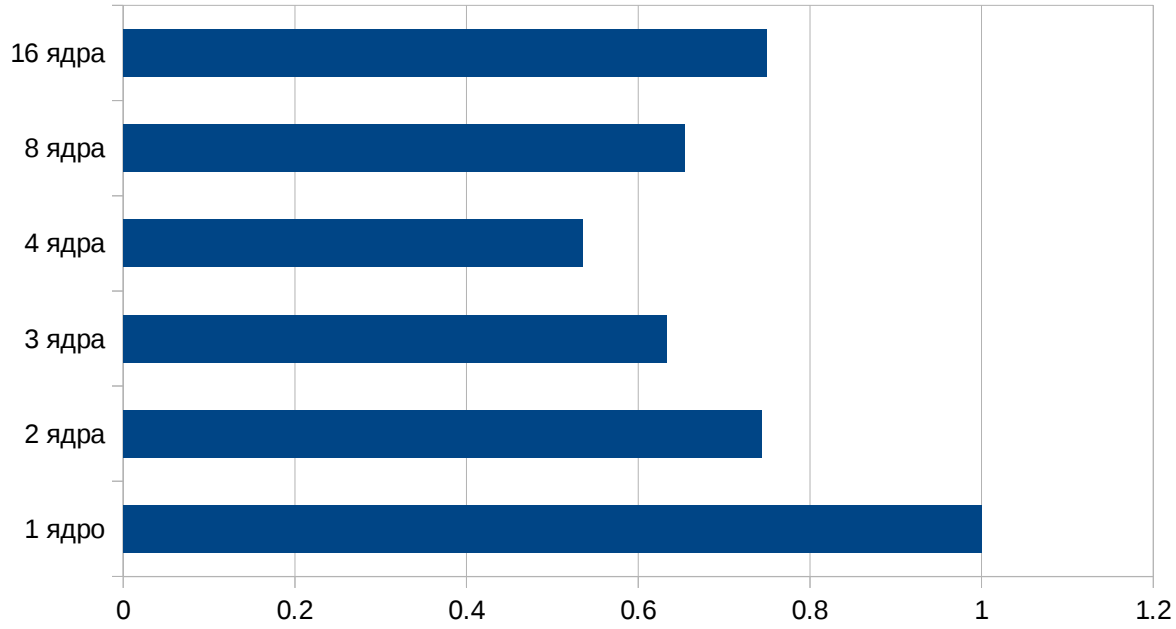
Следните графики показват какви скорости и ускорения може да се очаква с гореспоменатия алгоритъм.



Фиг. 13 *Показва средното време за изчисление*



Фиг. 14 *Показва ускорението*



Фиг. 15 Показва ефективността

3. Заключителни думи

Въпреки че последния алгоритъм дава резултати, които биха могли да се интерпретират като задоволителни, то тези алгоритми са далече от идеални. Силно зависими са от хардуер и софтуер който е пуснат на заден план. Също така е важно да се отбележи че характера на задачата не е паралелен и не бива да се очакват зашеметяващи резултати(или поне така мисли автора). Важно за разчитането на графиките е, че не всички резултати са в минути, някои са в секунди.