Дата: 01.06.2019

Проект
по Системи за Паралелна Обработка

Пресмятане на Pi

Сходящ ред на Ramanujan

Изготвил:

Владимир Ананиев, 81459
КН, 3 Курс, 1 Поток, 2 Група

Ръководител:

ас.. Христо Христов

_____

Съдържание

Описание на задачата

Целта на проекта е да реализира паралелен алгоритъм за пресмятане на числото π със зададена точност (значещи цифри след десетичната запетая), използвайки множество нишки. За целта се използва следният безкраен ред на индийския математик *Srinivasa Ramanujan*:

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!(1103 + 26390k)}{(k!)^4 396^{4k}}$$

Програмата е реализирана на програмният език Scala.

Упътване за употреба

Програмата се изпълнява през командния ред чрез командата
java -jar parallel-pi.java

Командни параметри

1. Параметър -p <число>, указващ броя на значещите цифри след десетичната запетая на резултата.
2. Параметър -t <число>, указващ броя нишки, които да се използват
3. Параметър -o <име на файл>, указващ името на файла, в който да бъде записан резултата.
4. Параметър -q, указващ “тих” режим на работа (без детайлни логове)

Резултат

Резултатът от програмата ще бъде записан във текстов файл, зададен като параметър на програмата. Ако не бъде изрично зададен такъв, резултатът ще бъде записан във файл по подразбиране на име output.txt

Описание на алгоритъма

Ако разгледаме сходящия ред, съобразяваме че не бихме имали полза да паралелизираме константния коефициент пред сумата. Пресмятането на членовете на сумата е независимо и там можем да се възползваме от множество процесори.

$$\sum_{k=0}^{\infty} \frac{(4k)!(1103 + 26390k)}{(k!)^4 396^{4k}}$$

Стратегията, която ще използваме е да разделим по равен брой членове на редицата за всяка нишка, която ще стартираме.

Например, ако имаме 3 нишки:

1. първата ще пресмята сумите на всички елементи кратни на 3.
2. втората ще пресмята сумите на елементите с остатък 1 при деление на 3.
3. третата ще пресмята сумите на елементите с остатък 2 при деление на 3.

Всяка нишка спира работа, когато членовете на редицата станат достатъчно малки, че не добавят стойност при зададената прецизност. Например, ако прецизността е 5, изчисленията приключват когато се достигне елемент със сума по-малка от 0.00001.

Анализ на резултатите

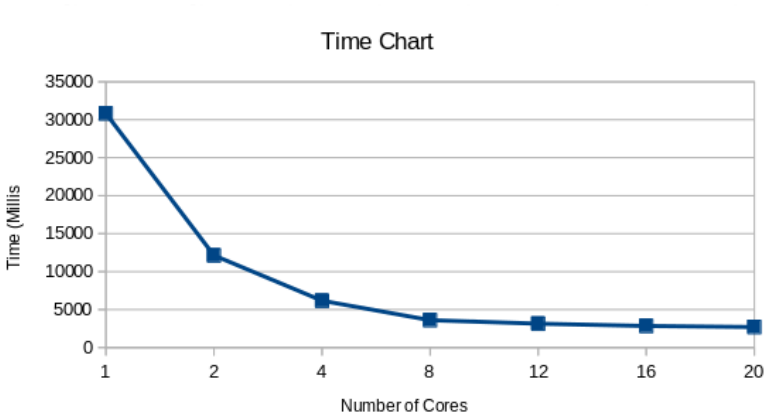
Тестовете са изпълнени на машина със следните параметри:

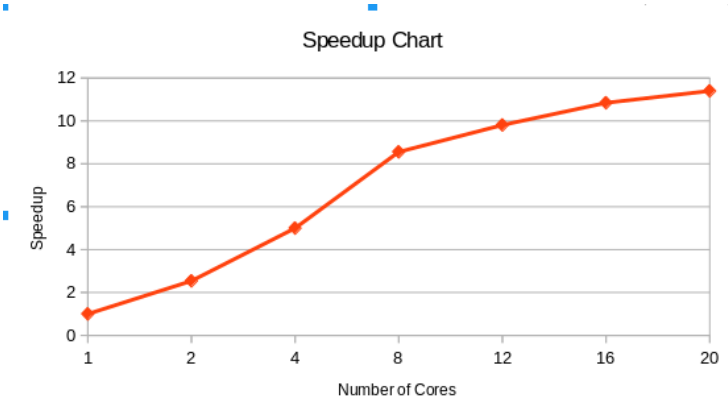
Architecture: x86_64

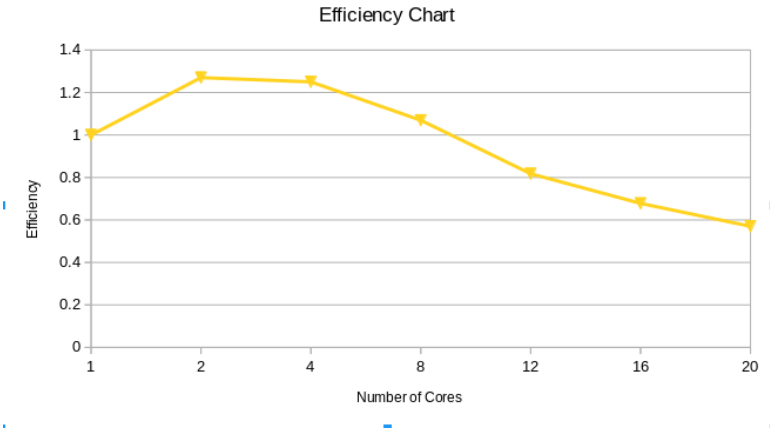
CPU(s): 32

Model Name: Intel(R) Xenon(R) CPU E5-2660 0 @ 2.20GHz

RAM: 62GB







Достигаме забързване от 11.4, използвайки 20 нишки.

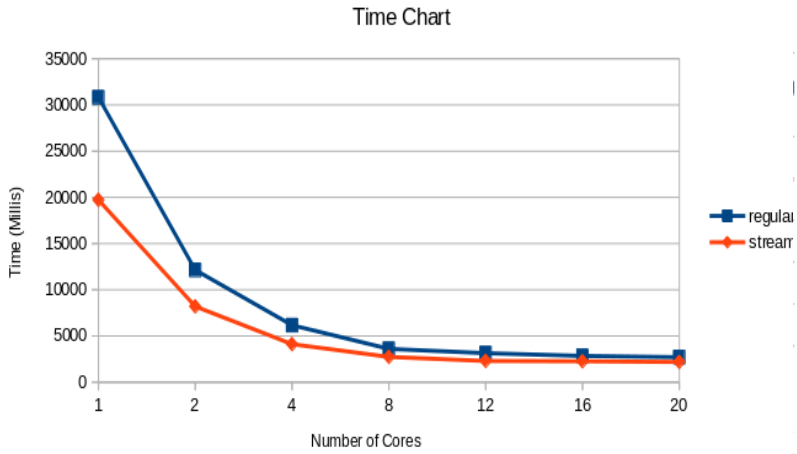
Интересно е да се забележи, че при 2, 4 и 8 нишки имаме ефективност над 1.

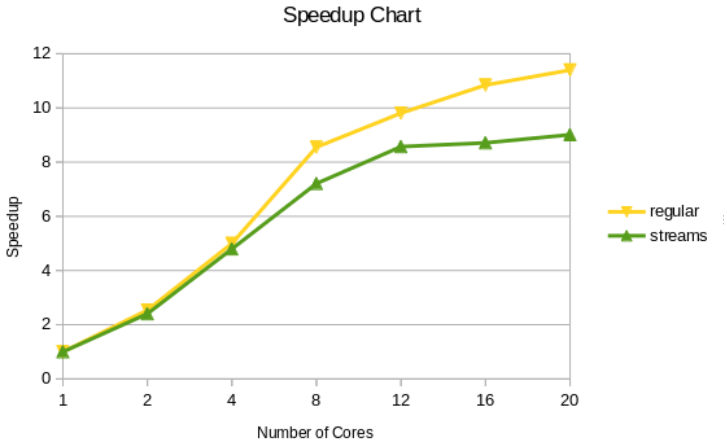
Пример за суперлинейност?

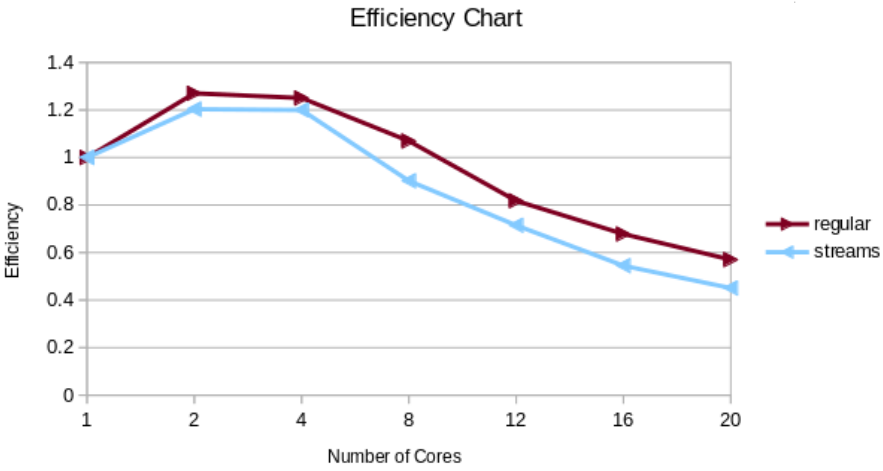
Оптимизация – Използване на lazy stream

Ако проследим изпълнението на програмата се забелязва, че най-времеемкото изчисление е това на факториел. Ще се опитаме да оптимизираме това изчисление като използваме lazy stream, който ни предоставя езика Scala. Той представлява immutable поток от данни,

който е споделен между всички нишки и всяка стойност се изчислява точно веднъж, след което стойността се запазва и не се изчислява при последващо достъпване. Така ще предотвратим повторното изяисляване на факториел и очакваме да забързаме програмата си.







Тук забелязваме, че времето което ни отнема за една нишка е значително по-малко.

Но за сметка на това забързването от добавяне на нови нишки е по-малко (имаме споделена памет и изчакване при достъпването ѝ.

Заключение

Успяхме да постигнем ускурение от над 11 при използване на 20 нишки. Оптимизачията на факториел чрез lazy stream е полезна при серийното изпълнение на програмата, но губи ефективността си изпълнение с повече нишки.