n(aughty) Body

Скалируемост при паралелна симулация на n тела в гравитационно поле

Проект по Системи за паралелна обработка

Николай Димов Паев, КН, 3-ти курс, 1-ви поток
ф.н.: 81983

Задачата за симулация на движението на n тела в гравитационно поле е често срещана в астрофизиката при моделирането на движенията на небесни тела - например астероиди, спътници, планети, звезди. Подобна задача се среща и при моделирането на разнообразни динамични системи като процесите в метеорологията.

Основната изчислителна сложност на задачата идва от взаимодействието на всяко тяло с всяко. Тази задача подлежи на паралелизация и в настоящия проект са представени две решения, които имплементират алгоритми с различни топологии - звезда и конвейер.

Постановката на задачата е следната: Разглеждат се голям брой тела, които са разпръснати в двумерно пространство и които имат следните характеристики: координати, вектор на скоростта и маса. Всяко тяло действа със сила на останалите и това взаимодействие се изчислява по уравнението на Нютон:

$$\sum_{i \neq n}^{N} \frac{G m_n m_i}{{r_{ni}}^2} \hat{r}_{ni}.$$

Съществуват различни решения на задачата. Редица от тях са описани в *On Distributed Gravitational N-Body Simulations* [1]. Там са представени така наречените йерархични апроксимиращи методи, какъвто е именно алгоритъма на Barnes-Hut, който често се използва симулация на голям брой тела (над 10 000). Той апроксимира действието на на далечни тела, като вместо да пресмята действието на всяко поотделно ги представя заедно. Така не се губи много точност и се намалява сложността на задачата ефективно от O(n²) до O(nlog n). Далечните тела за всяко тяло обаче са различни и това прави алгоритъма по-сложен. В настоящия проект, проявяваме интерес към ускорението при паралелизацията на някой последователен алгоритъм и затова метода на Barnes-Hut не е разгледан.

Последователен алгоритъм би се състоял от следните стъпки:

- За всяко тяло и всяко друго се изчислява действието на второто върху първото и това действие се прибавя към вектора на приложените сили на първото тяло.
- Актуализират се позициите на всяко едно тяло на базата на вектора на силата му.
- Като подготовка за следващата итерация се зануляват векторите за силата на всяко тяло.

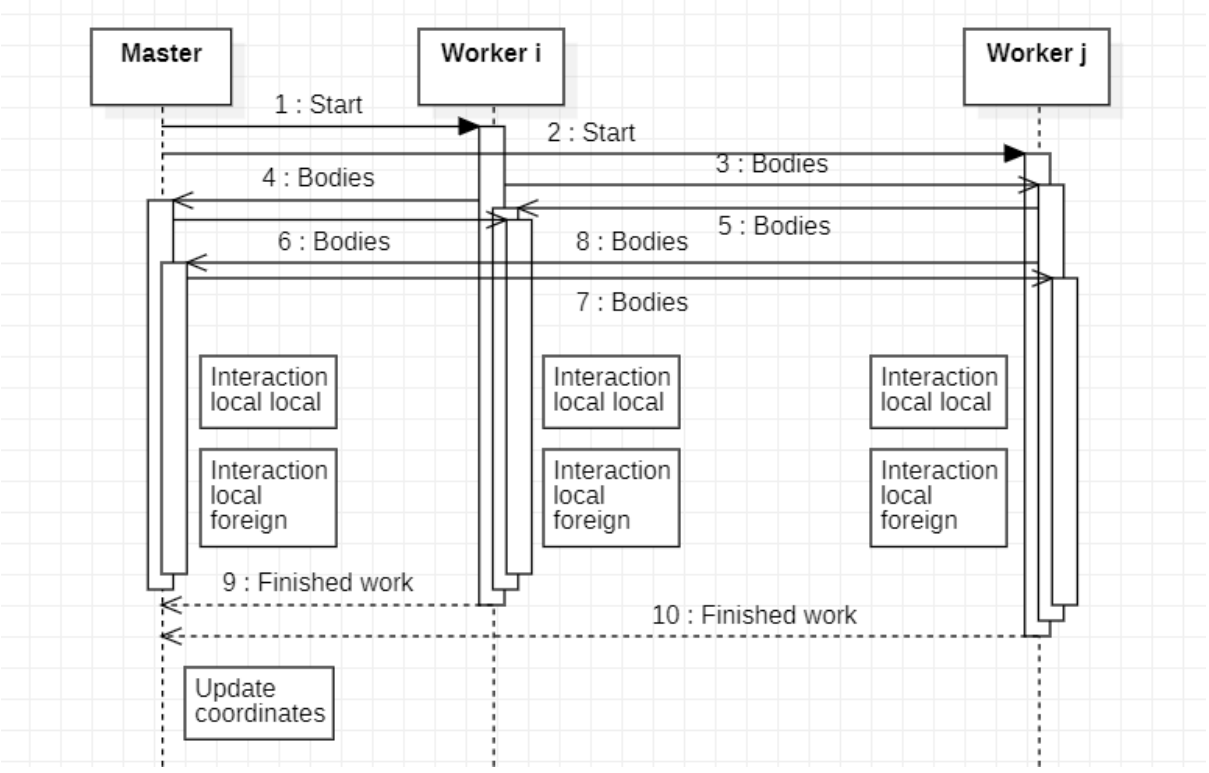
Актуализирането на позициите на телата не може да се случва едновременно с изчисляването на взаимодействията, тъй като това би довело до състезателен достъп - възможна е ситуация, в която някой процес ще достъпва позицията на някое тяло, докато друг я променя. Така, че изчисляванията на местоположенията и подготовката за следващата стъпка ще се случват последователно. Това не е проблем, тъй като тези операции имат линейна сложност спрямо броя на телата, докато операцията изчисляване на взаимодействието - има квадратична сложност. Ще паралелизираме точно нея. Така очакваме с увеличаване на броя на телата - времето, в което се изпълнява последователен код да расте по-бавно, отколкото времето, за което се изпълнява паралелен код, и според закона на Амдал [2], ускорението ще расте.

Тези разсъждения дават идея за следния паралелен алгоритъм с топология SPMD (който по-надолу е наречен “директен”):

Главен процес стартира p-1 на брой worker-процеси и създава p комуникационни канала - по един за всеки от тях и за себе си. Всеки работник отговаря за равен брой тела, като разпознава своите по номера си и ги изпраща по каналите на всеки друг процес. Главния процес дава сигнал на работниците за стартиране на следваща итерация. Всеки Worker-процес, а също и главния, за да не стои без работа, в рамките на една итерация изпълнява следния алгоритъм:

- Изчислява взаимодействията на собствените си тела едно с друго.
- Прочита телата получени на своя канал и прилага действието им върху своите тела
- Изпраща своите тела по каналите на всеки друг процес - подготвя следващата итерация.

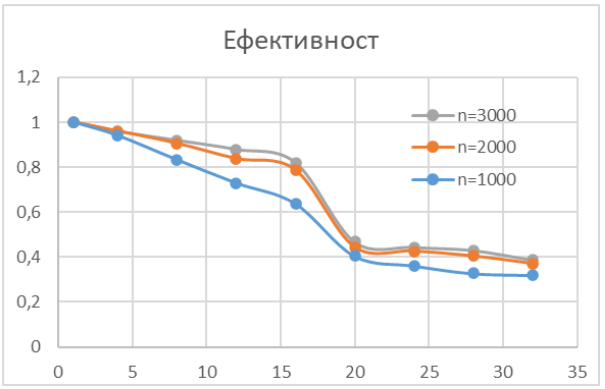
Главния процес след като свърши със своите изчисления изчаква всички работници и изпълнява последователния код за актуализиране на координатите на телата. След това дава сигнал за стартиране на следващата итерация.
(По каналите се предават указатели към телата, не самите тела, т.е. те служат по-скоро за синхронизация.)
Sequence диаграма на една итерация:



Алгоритъма, както и следващия по-долу, са реализирани на езика Go, който предоставя удобни примитиви за конкурентно програмиране. Тестовете (предоставени за справка в края на документа) са проведени на хардуер със следните характеристики:

CPU: Intel Xeon CPU E5-2660 2.20GHz
Ядра / хардуерен паралелизъм: 16
L1 d-cache на ядро: 32K
RAM: 64GB
OS: CentOS Linux 7

Ускорение и ефективност при различна изчислителна сложност:



Забелязва се, че с увеличаване на изчислителната сложност на задачата, ускорението расте. Това е очаквано , защото операциите за изчисление на взаимодействието от тип всяко с всяко растат квадратично спрямо броя на телата, докато синхронизационния свръхтовар не зависи от изчислителната сложност.

Задачата за симулация на n тела е задача, в която се извършва изчисление върху всички двойки от някакво множество обекти. В *A Generic All-Pairs Cluster-Computing Pipeline and its Applications* [3] е предложен паралелен алгоритъм с топология MPMD (конвейер) за всевъзможни задачи от тип “всички двойки”. Той е приложим за задачи, където обектите поддържат операциите:

- interact - взаимодействието между два обекта (симетрично - след изпълнение на операцията се променя състоянието и на двата обекта)
- integrate - актуализиране на състоянието на цялата система

В задачата n-Body, операцията interact е взаимодействието между две тела, а integrate - актуализирането на позициите на телата след прилагане на силите.

Конвейерът работи по следния начин:

- Главен процес изпраща едно по едно телата по конвейера.
- Всеки възел получава тела от левия си комуникационен канал.
- Запазва част от тях в локален масив и ги приема за свои.
- Изпълнява операцията interact между всяка двойка от своите тела.
- За всяко друго тяло, което пристигне на левия канал изпълнява interact между него и своите, след това го предава надясно.
- Накрая изпраща своите тела към главния процес (а не надясно по конвейера), който извършва операцията integrate

Това обаче води до дисбаланс в работата на отделните възли - операциите ще намаляват линейно от първия към последния възел, тъй като се реализира обхождане от тип:

```
for i from 0 to n do
    for j from i to n-1 do
        interact(body[i], body[j])
```

Такъв дисбаланс ще навреди на ускорението, което очакваме. В *A Generic All-Pairs Cluster-Computing Pipeline and its Applications* [3], а също и по-конкретно в *The N-Body Pipeline* [4], проблемът е решен чрез “сгъване” на конвейера, на няколко пъти върху p процеса.

В предложения тук алгоритъм, е заобиколена тази трудност. Операцията interact не е нужно да е симетрична, т.е. да променя състоянието и на двете тела, а може да е реализирана като метод към обекта тяло и да променя само него чрез информацията за другото тяло. Тогава алгоритъма може да придобие този вид:

```
for i from 0 to n do
    from j from 0 to n do
        if i != j then
            body[i].addForce(body[j])
```

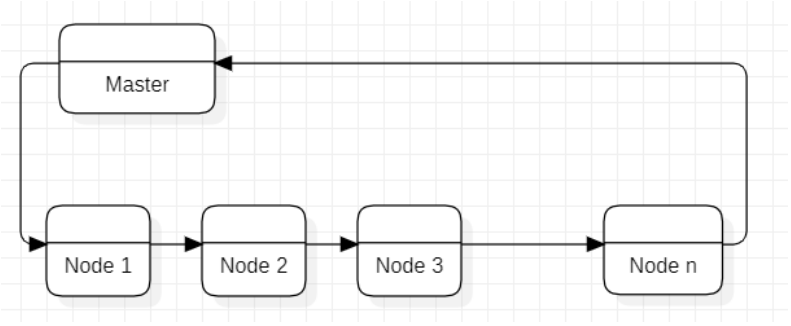
Забелязва се, че броя на операциите няма да се промени, т.к. симетричната операция interact по-горе крие в себе си двете операции:

```
body[i].addForce(body[j])
body[j].addForce(body[i])
```

С тази лека промяна натоварването на всеки възел ще бъде еднакво и той ще изпълнява следните действия:

- Получаване на локалните си тела. Смятане на взаимодействието помежду им.
- За всяко пристигнало друго тяло се добавя влиянието му към локалните, като се променя състоянието само на локалните тела и след това тялото се изпраща надясно към следващия възел.
- Накрая след като преминат всички чужди тела, възела изпраща и локалните си тела по конвейера. Така никой друг възел няма да ги вземе тях за локални.

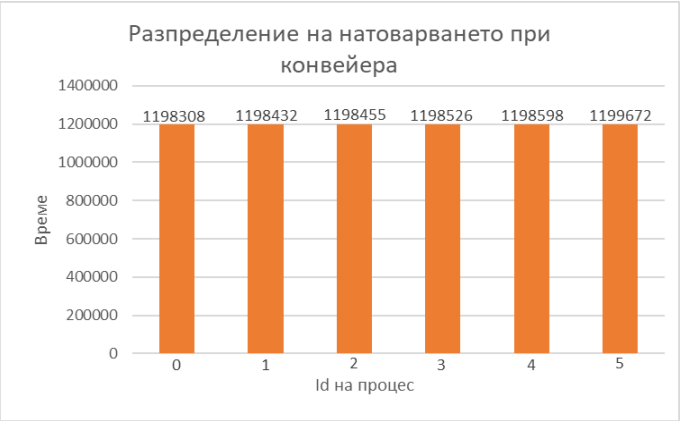
Главния процес получава телата едно по едно от изхода на конвейера и изпълнява операцията за актуализиране на позицията им веднага след като пристигнат. Така той не чака всички процеси да завършат, за да започне актуализацията.



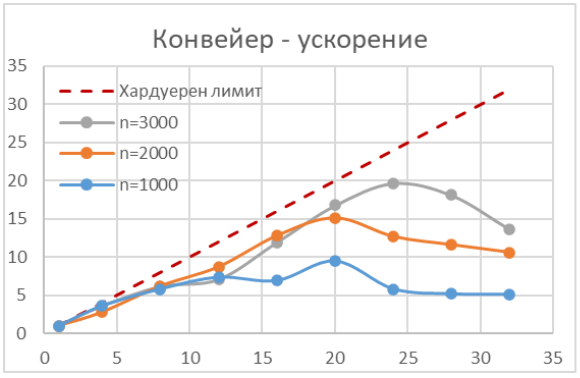
Конвейерът решава по лесен начин и въпроса със синхронизацията. Състояния на едновременно писане, или едновременно писане и четене от различни процеси са

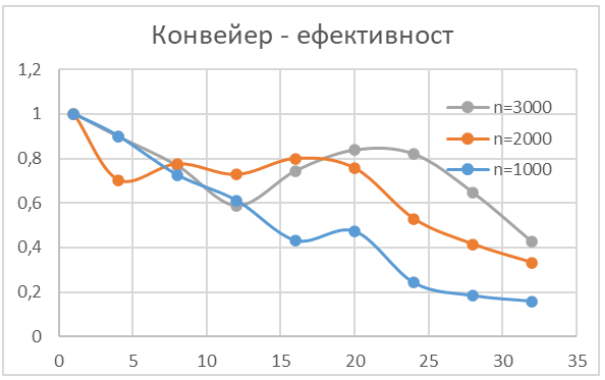
невъзможни. Всеки възел, докато променя данните за своите тела (при взаимодействието с други) ги държи в локална памет - те не се движат по конвейера и друг процес няма да извърши четене от техните полета.
От друга страна, когато мастър възела започне актуализирането на координатите на някое тяло, е гарантирано, че никой процес няма да пише или чете по полетата на това тяло, т.к. то е напуснало конвейера.

Натоварването е добре разпределено, както е видимо от графиката. Данните показват време на работа в микросекунди на всички 6 възела при p = 6, n = 2000 и 1000 стъпки.

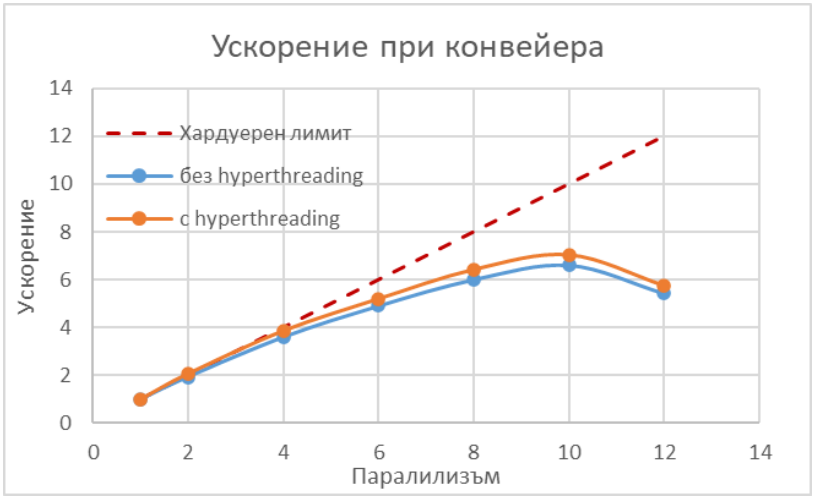


Тестовете отново показват, че, за да се наблюдава добро ускорение е необходимо да имаме висока изчислителна сложност. Освен това конвейерния алгоритъм има тенденция да продължава да демонстрира ускорение и при паралелизъм по-голям от броя на реалните ядра на тестовия хардуер.



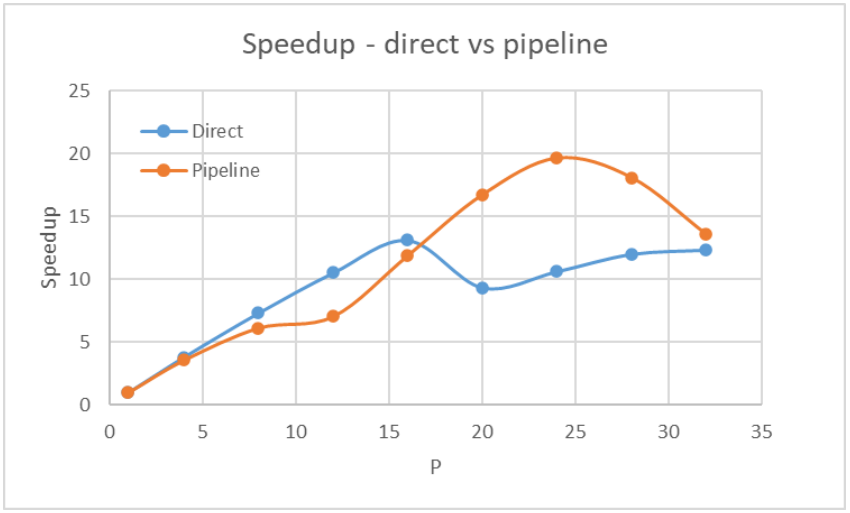


Постига се ускорение от 20 при паралелизъм 24, въпреки, че тестовата машина има хардуерен паралелизъм от 16. Това ни навежда на мисълта, че отговорник за това явление може да е технологията на Intel - hyperthreading. За да се провери тази хипотеза са направени последователно тестове с и без hyperthreading на машина с хардуерен паралелизъм 6 при 2000 тела.

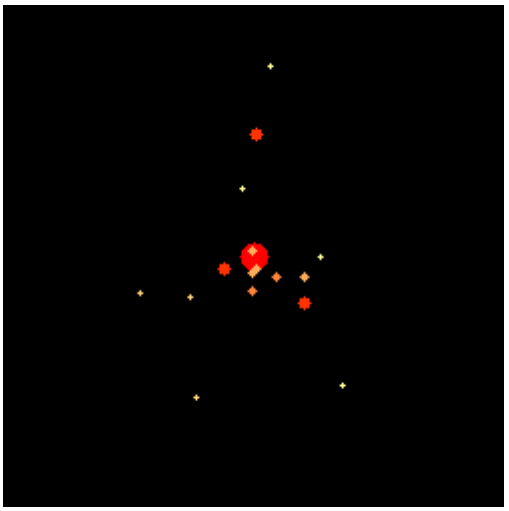


Забелязва се, че технологията hyperthreading не допринася много за паралелизма, а и по-важно - не тя е отговорна за високото ускорение след като сме преминали физическия брой ядра. Възможно е причината за това явление да е адаптивност към кеша за данни при някакво съотношение на брой процеси и брой тела.

Сравнение на ускорението при двете решения за n = 3000:



Като заключение, директния алгоритъм дава по-добро ускорение при малък паралелизъм, но конвейерния изглежда по-скалируем.



Към програмата е добавена и функционалност за генериране на анимации под формата на GIF:

Планетите се генерират като тела със случайни параметри, докато централното тяло - Слънцето е с фиксирана голяма маса и нулев вектор на скоростта.
Размера и оцветяването на телата, зависи от масата им (с изключение на Слънцето, чиито

размер на картинката е малко по-голям от този на планетите, но масата е значително по-голяма)

Кода може да бъде намерен тук:
https://github.com/NikolayDPaev/nBody

Източници:
[1]: On Distributed Gravitational N-Body Simulations, Alexander Brandt, Department of Computer Science University of Western Ontario, London, Canada Email: abrandt5@uwo.ca https://arxiv.org/pdf/2203.08966.pdf

[2]: Amdahl's law in the context of heterogeneous many-core systems – a survey Mohammed A. Noaman Al-hayanni,Fei Xia,Ashur Rafiev,Alexander Romanovsky,Rishad Shafik,Alex Yakovlev https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/iet-cdt.2018.5220

[3]: A GENERIC ALL-PAIRS CLUSTER-COMPUTING PIPELINE AND ITS APPLICATIONS A. RADENSKI Computer Science Dept. Winston-Salem State University, Winston-Salem, NC 27110, USA E-mail:radenski@computer.org B. NORRIS Computer Science Dept., University of Illinois at Urbana-Champaign, 1304 W. Springfield Ave., Urbana, Illinois 61801, USA W. CHEN Computer Science Dept. Winston-Salem State University, Winston-Salem, NC 27110, USA http://www1.chapman.edu/~radenski/research/papers/generic.pdf

[4]: The N-Body Pipeline Per Brinch Hansen Syracuse University, School of Computer and Information Science, pbh@top.cis.syr.edu https://surface.syr.edu/cgi/viewcontent.cgi?article=1106&context=eecs_techreports

Тестове:

#	p	n	*algorithm*	$T_p^{(1)}$	$T_p^{(2)}$	$T_p^{(3)}$	$T_p = \min(T_p^{(i)})$	$S_p = T_1/T_p$	$E_p = S_p/p$
1	1	1000	direct	***29738362***	***29911069***	***29781738***	**29738362**	1	1
2	4	1000	direct	*8161552*	*8120352*	*7907696*	**7907696**	3,760686046	0,940171511
3	8	1000	direct	*4630375*	*4609686*	*4464210*	**4464210**	6,661506067	0,832688258
4	12	1000	direct	*3507656*	*3497184*	*3397939*	**3397939**	8,751882244	0,72932352
5	16	1000	direct	*2977646*	*2916150*	*3799267*	**2916150**	10,1978163	0,637363519
6	20	1000	direct	*3806767*	*3800194*	*3676173*	**3676173**	8,08948926	0,404474463
7	24	1000	direct	*3565472*	*3543638*	*3431261*	**3431261**	8,666890103	0,361120421
8	28	1000	direct	*3356228*	*3350534*	*3247326*	**3247326**	9,157799987	0,327064285
9	32	1000	direct	*2979716*	*2977371*	*2911027*	**2911027**	10,21576303	0,319242595
10	1	1000	pipeline	***30086744***	***31574658***	***29838035***	**29838035**	1	1
11	4	1000	pipeline	*13524762*	*10385197*	*8283488*	**8283488**	3,602110005	0,900527501
12	8	1000	pipeline	*6010250*	*5140375*	*5993905*	**5140375**	5,804641685	0,725580211
13	12	1000	pipeline	*4852637*	*4150457*	*4066678*	**4066678**	7,337201273	0,611433439
14	16	1000	pipeline	*4908558*	*4605081*	*4316875*	**4316875**	6,911952512	0,431997032
15	20	1000	pipeline	*3540888*	*3154593*	*3195985*	**3154593**	9,458600523	0,472930026
16	24	1000	pipeline	*5614130*	*5190428*	*5140238*	**5140238**	5,804796393	0,241866516
17	28	1000	pipeline	*6377808*	*5999724*	*5776612*	**5776612**	5,165317491	0,184475625
18	32	1000	pipeline	*6235354*	*5851823*	*5902943*	**5851823**	5,098929855	0,159341558

19	1	2000	direct	***118990957***	***119408359***	***118996370***	**118990957**	1	1
20	4	2000	direct	*30980645*	*31414869*	*31173967*	**30980645**	3,840816	0,960204
21	8	2000	direct	*16640618*	*17472778*	*16429166*	**16429166**	7,242665696	0,905333212
22	12	2000	direct	*12086603*	*12479052*	*11845729*	**11845729**	10,04505143	0,837087619
23	16	2000	direct	*9559611*	*15213360*	*9447734*	**9447734**	12,59465571	0,787165982
24	20	2000	direct	*13563568*	*13557693*	*13396340*	**13396340**	8,882348238	0,444117412
25	24	2000	direct	*11755269*	*11803838*	*11646591*	**11646591**	10,21680567	0,425700236
26	28	2000	direct	*10604261*	*10708643*	*10516861*	**10516861**	11,31430348	0,404082267
27	32	2000	direct	*10052284*	*11401678*	*10122300*	**10052284**	11,83720605	0,369912689
28	1	2000	pipeline	***120748243***	***144154751***	***119386740***	**119386740**	1	1
29	4	2000	pipeline	*51927386*	*49202747*	*42444303*	**42444303**	2,81278597	0,703196493
30	8	2000	pipeline	*22694191*	*20084655*	*19237697*	**19237697**	6,205874851	0,775734356
31	12	2000	pipeline	*15351736*	*14382668*	*13682213*	**13682213**	8,725689331	0,727140778
32	16	2000	pipeline	*10799446*	*9330835*	*9376493*	**9330835**	12,79486134	0,799678834
33	20	2000	pipeline	*8823910*	*7950927*	*7899929*	**7899929**	15,11238139	0,755619069
34	24	2000	pipeline	*9894705*	*9539013*	*9411028*	**9411028**	12,68583411	0,528576421
35	28	2000	pipeline	*10412245*	*10439789*	*10283411*	**10283411**	11,60964392	0,41463014
36	32	2000	pipeline	*11839763*	*11809646*	*11243358*	**11243358**	10,61842378	0,331825743

37	1	3000	direct	***267818351***	***267973860***	***267642184***	**267642184**	1	1
38	4	3000	direct	*69789507*	*71422095*	*76755913*	**69789507**	3,834991756	0,958747939
39	8	3000	direct	*36654506*	*37725615*	*36428785*	**36428785**	7,346997272	0,918374659
40	12	3000	direct	*25777934*	*26901531*	*25404183*	**25404183**	10,53535884	0,87794657
41	16	3000	direct	*20576453*	*32319827*	*20400681*	**20400681**	13,11927695	0,819954809
42	20	3000	direct	*28763819*	*29513981*	*28619784*	**28619784**	9,351649335	0,467582467
43	24	3000	direct	*25293949*	*25476047*	*25144129*	**25144129**	10,64432115	0,443513381
44	28	3000	direct	*22373084*	*22545694*	*22267224*	**22267224**	12,01955771	0,429269918
45	32	3000	direct	*21896382*	*22608959*	*21650586*	**21650586**	12,36189099	0,386309093
46	1	3000	pipeline	***338136506***	***337977223***	***270391111***	**270391111**	1	1
47	4	3000	pipeline	*81057442*	*81012056*	*75317505*	**75317505**	3,590016836	0,897504209
48	8	3000	pipeline	*53855233*	*55705379*	*44028239*	**44028239**	6,141311057	0,767663882
49	12	3000	pipeline	*40267958*	*38984476*	*38254401*	**38254401**	7,068235391	0,589019616
50	16	3000	pipeline	*23616652*	*24540444*	*22683225*	**22683225**	11,92031164	0,745019477
51	20	3000	pipeline	*17764761*	*16133114*	*17703616*	**16133114**	16,76000746	0,838000373
52	24	3000	pipeline	*14646215*	*13749246*	*14055539*	**13749246**	19,66588648	0,819411937
53	28	3000	pipeline	*15296713*	*15524764*	*14933400*	**14933400**	18,10646678	0,646659528
54	32	3000	pipeline	*21570251*	*23918598*	*19866005*	**19866005**	13,61074413	0,425335754