Flop count vs. Efficiency

Edilbert Christhuraj and Sadulla Aghayev

February 2, 2018

RWTH Aachen

Problem Statement

"In the design of a numerical algorithms, one typically tends to minimize the number of floating point operations, with the intention of minimizing the execution time. The underlying assumption, which unfortunately does not hold in practice, is that all flops cost the same.

The project is an investigation of the difference between flop count and execution time, plus a study of sensitivity to perturbations".

Prof. Dr. Paolo Bientinesi

Hight Performance and Automatic Computing

We need to...

(i) investigate difference between flop count and execution time

(ii) verify the statement - "All flops do not cost the same"

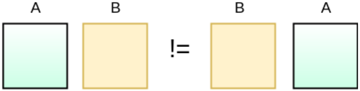
(iii) carry out a perturbation analysis

Additional requirement

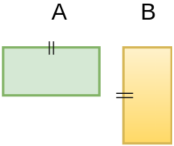
use an optimized BLAS library function to perform computations

Recall

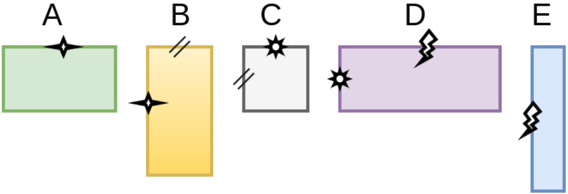
Matrix multiplication is associative but not always commutative



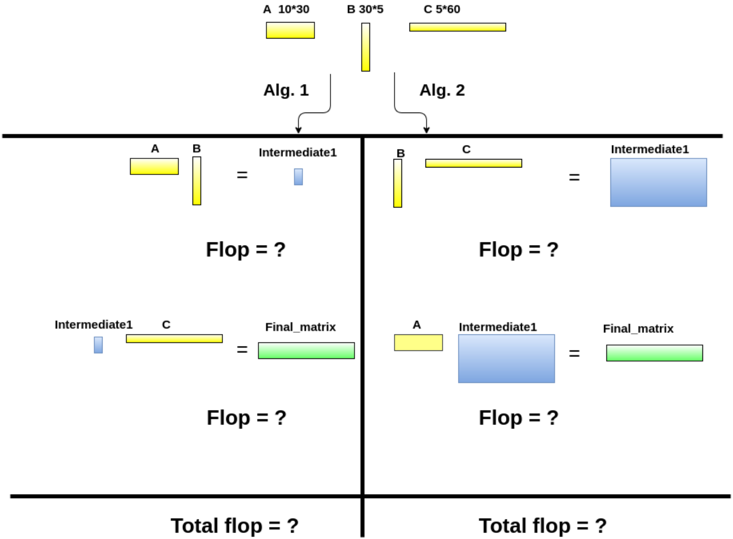
Dimensions must agree



Matrix chain multiplication

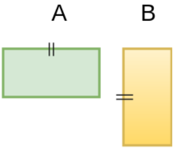


A simple 3 matrices chain



FLOP calculation

FLOP - Floating Point Operation



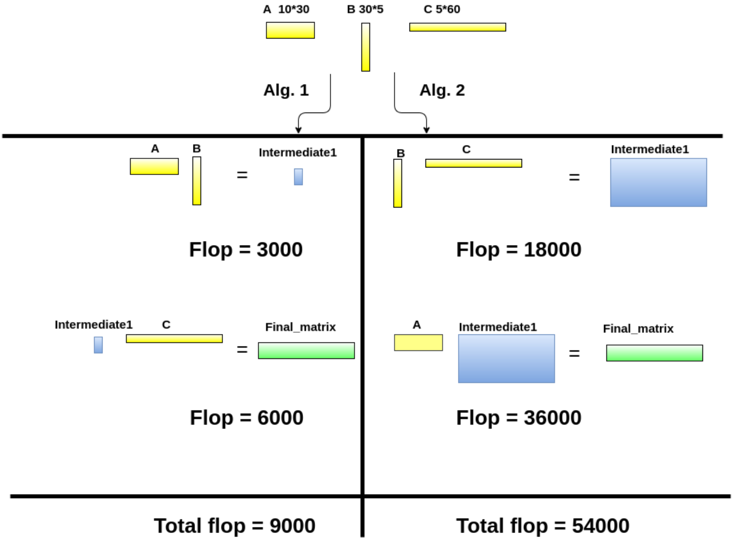
A $(m * n)$ and B $(n * k)$

```
for(a=0; a < m; ++a)
{
    for(b = 0; b < k; ++b)
    {
        for(c = 0; c < n; ++c)
        {

            result[a][b] = result[a][b] + A[a][c] * B[c][b];

        }
    }
}
```

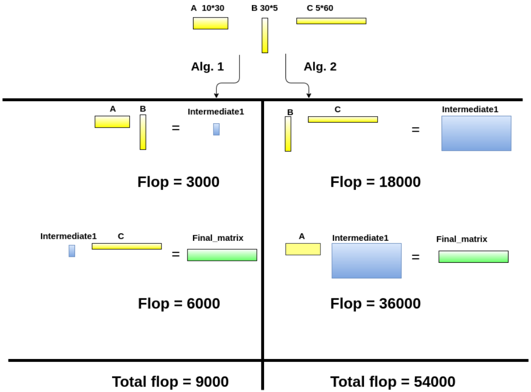
$$\text{Flop} = 2 * m * n * k$$

Example: A $(10 * 20)$ and B $(20 * 4)$

$$Flop = 2 * 10 * 20 * 4 = 1600$$


A 10*30
B 30*5
C 5*60
Alg. 1
Alg. 2
A
B
=
Intermediate1
Flop = 3000
B
C
=
Intermediate1
Flop = 18000
Intermediate1
C
=
Final_matrix
Flop = 6000
A
Intermediate1
=
Final_matrix
Flop = 36000
Total flop = 9000
Total flop = 54000

A simple 3 matrices chain

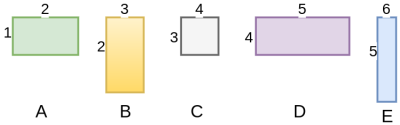


$$(A * B) * C = A * (B * C)$$

$$\text{Flop}(Alg.1) \neq \text{Flop}(Alg.2)$$

and execution time?

Catalan number and Parenthesization



possibility-1 (((A*B)*C)*D)*E

possibility-2 A*(B*(C*(D*E)))

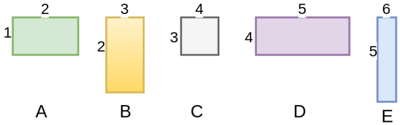
how many possibilities in total?

P_n parenthesization: $P_n = C_{n-1}$

n^{th} Catalan number : $C_n = \frac{1}{n+1}\binom{2n}{n}$

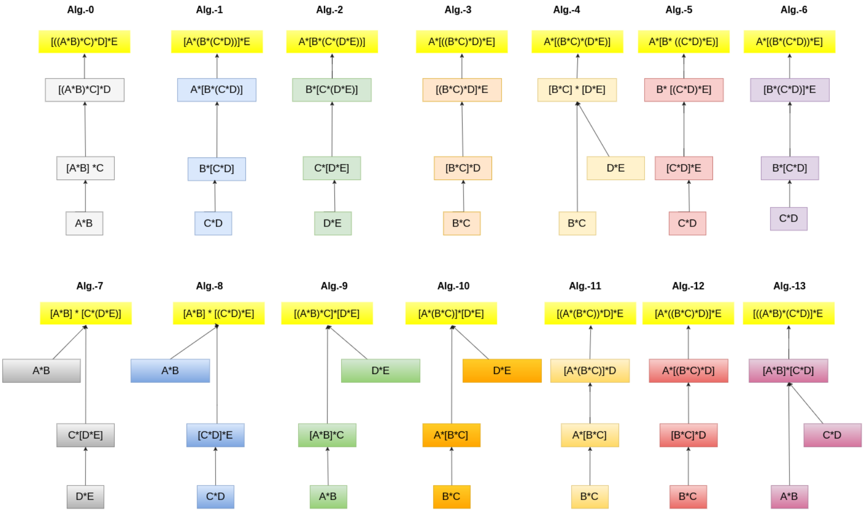
n	2	3	4	5	6	7	8	9
P_n	1	2	5	14	42	132	429	1430

5 Matrices chain



Alg.-0 (((A*B)*C)*D)*E
Alg.-1 (A*(B*(C*D)))*E
Alg.-2 A*(B*(C*(D*E)))
Alg.-3 A*(((B*C)*D)*E)
Alg.-4 A*((B*C)*(D*E))
Alg.-5 A*(B*((C*D)*E))
Alg.-6 A*((B*(C*D))*E)
Alg.-7 (A*B)*(C*(D*E))
Alg.-8 (A*B)*((C*D)*E)
Alg.-9 ((A*B)*C)*(D*E)
Alg.-10 (A*(B*C))*(D*E)
Alg.-11 ((A*(B*C))*D)*E
Alg.-12 (A*((B*C)*D))*E
Alg.-13 ((A*B)*(C*D))*E

5 Matrices chain



Quantities that we are interested in are :

(i) **Flop** - How many floating point operations does each algorithm perform?
min_flop_alg - Algorithm which performs minimum flop

(ii) **Execution time (in Seconds)** - How much time does each algorithm take to execute?
min_time_alg - Algorithm which takes minimum time to execute

(iii) **Deviation (in %)** $= \frac{t_{min\_flop\_alg} - t_{min\_time\_alg}}{t_{min\_time\_alg}} * 100$
How much percentage is the "*min_time_alg*" faster than the "*min_flop_alg*" ?
$t_{min\_flop\_alg}$ - time for minimum flop algorithm
$t_{min\_time\_alg}$ - time for minimum time algorithm

Pseudo algorithm

main {

Timer start

Alg. 0

Timer finish

Timer start

Alg. 1

Timer finish

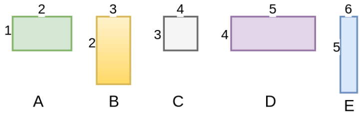
...

Timer start

Alg. 13

Timer finish

}



Input : Matrix sizes[1, 2, 3, 4, 5, 6]

output :

time_0	flop_0
time_1	flop_1
time_2	flop_2
.	.
.	.
time_12	flop_12
time_13	flop_13

Deviation

Additional information

OpenBLAS

(i) OpenBLAS is an optimized BLAS library based on GotoBLAS2
(ii) version - 0.2.20
(iii) cblas_dgemm(.....)

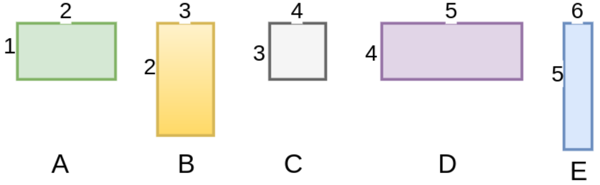
Hardware details

(i) RWTH Clutser - MPI_S
(ii) https://doc.itc.rwth-aachen.de/display/CC/Hardware+of+the+RWTH+Compute
(iii) tested only on one node

Programming environment

implementation in C and gcc compiler v4.0.2

Test case



1^{st} Chain 130 700 383 1340 193 900

2^{nd} Chain 376 561 477 532 425 590

Algorithm	**Time** (Seconds)	**Flop**	
0	0.051306	315,546,400	
1	0.065909	381,877,520	
2	0.247766	2,035,692,000	
3	0.180365	1,487,556,000	
4	0.361984	3,036,224,000	
5	0.118680	977,537,120	min_flop_alg-0
6	0.088250	708,569,520	min_time_alg-13
7	0.185151	1,548,640,000	Deviation-21.2%
8	0.061688	490,485,120	
9	0.122389	982,219,200	
10	0.215232	1,741,464,000	
11	0.131261	1,074,791,200	
12	0.140059	1,160,864,000	
13	0.042321	332,189,860	

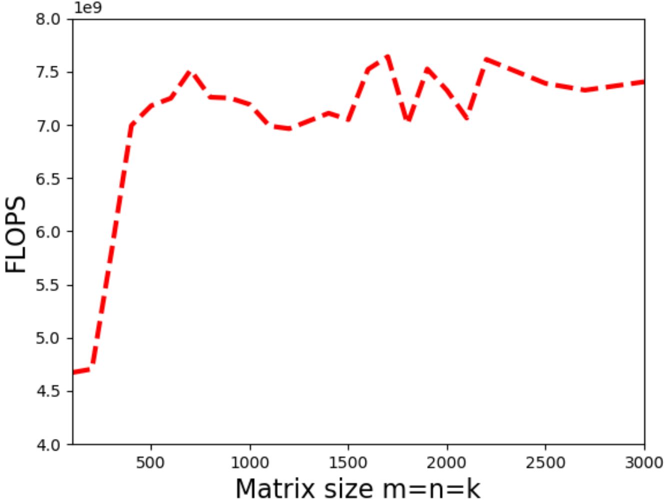
2^{nd} Chain 376 561 477 532 425 590 Sample-1

Algorithm	Time (Seconds)	Flop	
0	0.101891	750,654,672	
1	0.099232	811,016,450	
2	0.140206	1,130,908,460	
3	0.130707	1,068,653,388	
4	0.139825	1,152,599,048	
5	0.126232	1,019,583,840	min_flop_alg-0
6	0.121548	973,402,830	min_time_alg-13
7	0.121522	979,107,824	Deviation-8.44%
8	0.110458	867,783,204	
9	0.112590	894,899,232	
10	0.124695	1,011,994,872	
11	0.106382	867,750,312	
12	0.112451	906,267,008	
13	0.093955	757,945,544	

All flops do not cost the same

Matrix	Flop	FLOPS $= \frac{Flop}{Second}$
$(100 * 100) * (100 * 100)$	2,000,000	4,672,897,196
$(200 * 200) * (200 * 200)$	16,000,000	4,705,882,352
$(300 * 300) * (300 * 300)$	54,000,000	5,819,592,628
$(400 * 400) * (400 * 400)$	128,000,000	6,994,535,519
..	...	...
..	...	...
..	...	...
$(1300 * 1300) * (1300 * 1300)$	4,394,000,000	7,036,896,462
$(1400 * 1400) * (1400 * 1400)$	5,488,000,000	7,109,296,363
$(1500 * 1500) * (1500 * 1500)$	6,750,000,000	7,048,975,235
...	...	...
$(3000 * 3000) * (3000 * 3000)$	54,000,000,000	7,404,131,916

All flops do not cost the same



Perturbation analysis

(i) **Problem**
Some of the quantities that we are interested in, such as execution times and deviation are not reproducible

Algorithm	Time (Seconds)	Flop	
0	0.051306	315,546,400	
1	0.065909	381,877,520	
2	0.247766	2,035,692,000	
3	0.180365	1,487,556,000	
4	0.361984	3,036,224,000	
5	0.118680	977,537,120	min_flop_alg-0
6	0.088250	708,569,520	min_time_alg-13
7	0.185151	1,548,640,000	Deviation-21.2%
8	0.061688	490,485,120	
9	0.122389	982,219,200	
10	0.215232	1,741,464,000	
11	0.131261	1,074,791,200	
12	0.140059	1,160,864,000	
13	0.042321	332189860	

Algorithm	Time (Seconds)	Flop	
0	0.042322	315,546,400	
1	0.054053	381,877,520	
2	0.255563	2,035,692,000	
3	0.187326	1,487,556,000	
4	0.381132	3,036,224,000	
5	0.120140	977,537,120	min_flop_alg-0
6	0.087574	708,569,520	min_time_alg-0
7	0.189211	1,548,640,000	Deviation-0.0%
8	0.061826	490,485,120	
9	0.151340	982,219,200	
10	0.274111	1,741,464,000	
11	0.178278	1,074,791,200	
12	0.181404	1,160,864,000	
13	0.045157	332,189,860	

1^{st} Chain 130 700 383 1340 193 900 Sample-3

Algorithm	Time (Seconds)	Flop	
0	0.055871	315,546,400	
1	0.056499	381,877,520	
2	0.261406	2,035,692,000	
3	0.206040	1,487,556,000	
4	0.387012	3,036,224,000	
5	0.131321	977,537,120	min_flop_alg-0
6	0.092505	708,569,520	min_time_alg-13
7	0.196337	1,548,640,000	Deviation-25.4%
8	0.066044	490,485,120	
9	0.128545	982,219,200	
10	0.226485	1,741,464,000	
11	0.137919	1,074,791,200	
12	0.146254	1,160,864,000	
13	0.044547	332,189,860	

2^{nd} Chain 376 561 477 532 425 590 Sample-1

Algorithm	Time (Seconds)	Flop	
0	0.090527	750,654,672	
1	0.096886	811,016,450	
2	0.133695	1,130,908,460	
3	0.126657	1,068,653,388	
4	0.137426	1,152,599,048	
5	0.122484	1,019,583,840	min_flop_alg-0
6	0.116337	973,402,830	min_time_alg-13
7	0.122186	979,107,824	Deviation-0.0%
8	0.113717	867,783,204	
9	0.109312	894,899,232	
10	0.123241	1,011,994,872	
11	0.105787	867,750,312	
12	0.110508	906,267,008	
13	0.091250	757,945,544	

2^{nd} Chain 376 561 477 532 425 590 Sample-2

Algorithm	Time (Seconds)	Flop	
0	0.101891	750,654,672	
1	0.099232	811,016,450	
2	0.140206	1,130,908,460	
3	0.130707	1,068,653,388	
4	0.139825	1,152,599,048	
5	0.126232	1,019,583,840	min_flop_alg-0
6	0.121548	973,402,830	min_time_alg-13
7	0.121522	979,107,824	Deviation-8.44%
8	0.110458	867,783,204	
9	0.112590	894,899,232	
10	0.124695	1,011,994,872	
11	0.106382	867,750,312	
12	0.112451	906,267,008	
13	0.093955	757,945,544	

2^{nd} Chain 376 561 477 532 425 590 Sample-3

Algorithm	Time (Seconds)	Flop	
0	0.102252	750,654,672	
1	0.114649	811,016,450	
2	0.145311	1,130,908,460	
3	0.129112	1,068,653,388	
4	0.139820	1,152,599,048	
5	0.129094	1,019,583,840	min_flop_alg-0
6	0.120107	973,402,830	min_time_alg-13
7	0.120611	979,107,824	Deviation 9.12%
8	0.107226	867,783,204	
9	0.110580	894,899,232	
10	0.123456	1,011,994,872	
11	0.105604	867,750,312	
12	0.110942	906,267,008	
13	0.093701	757,945,544	

Perturbation analysis

(i) **Problem**

Some of the quantities that we are interested in, such as execution times and deviation are not reproducible

(ii) **Our attempts**

Cache flushing, different approaches for time measurements, iterations etc.

no improvement

(iii) **Remedy**

It is the nature of the problem.

Execution time depends not only on inputs but also on parameters such as system temperature, power supply etc.

Given Problem is based on empirical statement, not based on any mathematical equation.

Goal of Perturbation analysis

Perturbation percentage	Deviation range	Deviation frequency	Flop difference
+15%			
+10%			
+5%			
+3%			
+1%			
130 700 383 1340 193 900	0.01-36 %	27%	16,643,460
-1%			
-3%			
-5%			
-10%			
-15%			

Perturbation percentage	**Deviation range**	**Deviation frequency**	**Flop difference**
+15%	0.01-29 %	25%	25,128,102
+10%	3.5-15 %	8%	21,006,788
+5%	0.01-68 %	30%	19,442,328
+3%	0.01-40 %	40%	18,752,952
+1%	0.01-18.5%	40%	16,653,820
130 700 383 1340 193 900	0.01-36 %	27%	16,643,460
-1%	0.01-21 %	26%	17,017,292
-3%	0.01-42 %	30%	15,064,266
-5%	0.01-80 %	34%	14,485,500
-10%	0.01-22 %	30%	11,569,284
-15%	0.01-18 %	28%	10,609,780

2^{nd} Chain   All elements perturbation

Perturbation percentage	Deviation range	Deviation frequency	Flop difference
+15%	0.01-15 %	46.5%	10,937,920
+10%	3.5-6%	48%	9,576,058
+5%	0.01-67 %	52.5%	8,632,016
+3%	0.01-20 %	43%	7,916,372
+1%	0.01-19 %	63%	7,619,508
376 561 477 532 425 590	0.01-22 %	40%	7,290,872
-1%	0.01-13 %	65%	6,960,192
-3%	0.01-20 %	47.5%	6,89,088
-5%	0.01-73 %	57%	6,083,796
-10%	0.01-34 %	43%	5,392,088
-15%	0.01-19 %	40%	4,552,094

1^{st} Chain Single element perturbation (5%)

%	Matrix chain	Deviation range	Deviation frequency	Flop difference
+	136 700 383 1340 193 900	0.01-25 %	53%	8,628,408
+	130 735 383 1340 193 900	0.01-67 %	33%	16,643,460
+	130 700 402 1340 193 900	0.01-24 %	30%	20,804,840
+	130 700 383 1407 193 900	0.01-74 %	46%	16,514,686
+	130 700 383 1340 202 900	2 - 21 %	26%	23,642,040
+	130 700 383 1340 193 945	1 - 40 %	36%	16,643,460
	130 700 383 1340 193 900	0.01-36 %	27%	16,643,460
-	123 700 383 1340 193 900	0.01-50 %	23%	26,414,454
-	130 665 383 1340 193 900	0.01-85 %	56%	16,643,460
-	130 700 363 1340 193 900	0.01-13 %	43%	9,263,060
-	130 700 383 1273 193 900	4 - 84 %	30%	16,772,234
-	130 700 383 1340 183 900	0.01-26%	56%	8,867,260
-	130 700 383 1340 193 855	0.01-39 %	36%	16,643,460

Single element perturbation (5%)

%	Matrix chain	Deviation range	Deviation frequency	Flop difference
+	394 561 477 532 425 590	0.01-11 %	23%	2,686,132
+	376 589 477 532 425 590	0.01-17 %	40%	7,290,872
+	376 561 500 532 425 590	0.01-49 %	50%	15,840,800
+	376 561 477 558 425 590	0.01- 5 %	23%	196,668
+	376 561 477 532 446 590	0.01-12%	30%	17,080,400
+	376 561 477 532 425 619	0.01-16 %	43%	7,290,872
	376 561 477 532 425 590	0.01-22 %	40%	7,290,872
-	357 561 477 532 425 590	0.01-24 %	26%	17,822,154
-	376 532 477 532 425 590	0.01-32 %	41%	7,290,872
-	376 561 453 532 425 590	0.01-11 %	26%	1,630,792
-	376 561 477 505 425 590	0.01-44 %	43%	14,657,930
-	376 561 477 532 403 590	0.01- 9 %	26%	2,964,824
-	376 561 477 532 425 560	0.01-31 %	43%	7,290,872

Conclusion

(i) we investigated the relation between flop count and execution time and we verified the statement "all flops do not cost the same".

(ii) we carried out a perturbation analysis and studied quantities such as Flop difference, deviation range and deviation frequency for perturbed matrix chains.

References

(i) link to OpenBLAS
`http://www.openblas.net/`

(ii) link to dissertation "Performance Modeling and Prediction for Dense Linear Algebra"
`https://arxiv.org/pdf/1706.01341.pdf`

(iii) link to our source code
`https://github.com/edilbert24/Sisc-Lab/tree/master/Final_Sisc_code`