In3030

Marius Madsen mariumh

Oblig2

This Program is constructed to take any n*n matrices and multiply them. It can either be calculated sequential or parallel, also whether you want to solve it the classical a*b way, a transposed * b or a * b transposed. However these three solutions have different calculation time compared to one another.

First of all is the sequential way to solve them all:

a*b is the common way to solve it on a piece of paper. The formula is given by

$$c[i][j] = \sum_{j=0}^{n}\sum_{k=0}^{n} a[i][k] * b[k][j]$$

this solution is not recommended because the way b is accessed is bad considered it is bad cache accessing. This solution will have a computing time of n^3.

a transposed *b, is also a way to solve it. Transposing means to change every rows of the matrix with the corresponding column.

$$c[i][j] = \sum_{j=0}^{n}\sum_{k=0}^{n} aT[k][i] * b[k][j]$$

This one is even worse considering it just is not b that has bad cache accessing, but now we use a transposed which is as bad. On top of that it has $n^3+n^{2.}$

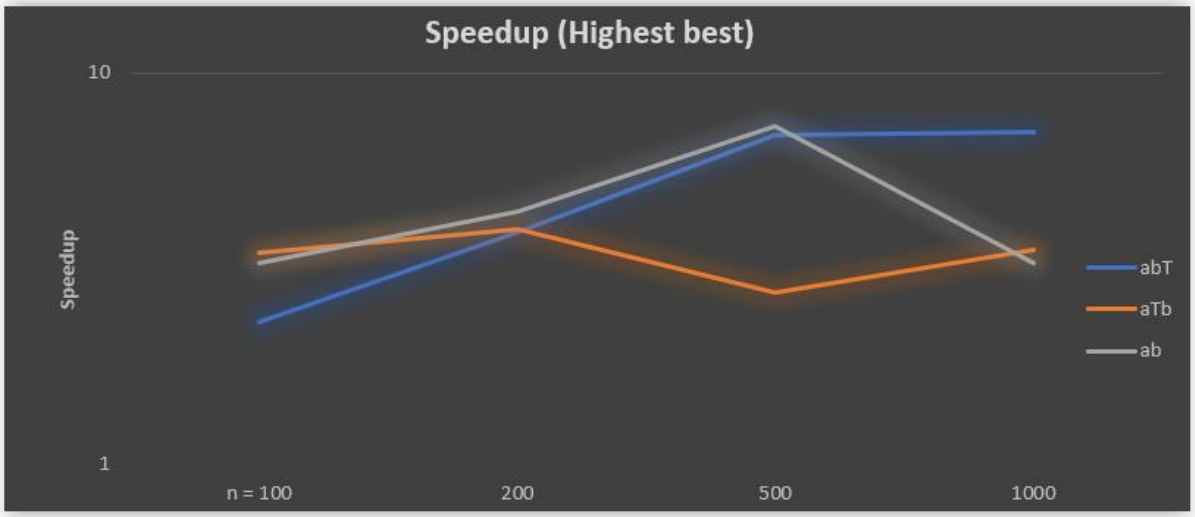
a*b transposed, is the final way it is solved, by transposing the b matrix instead, which will avoid the bad cache accessing. The formula is given by

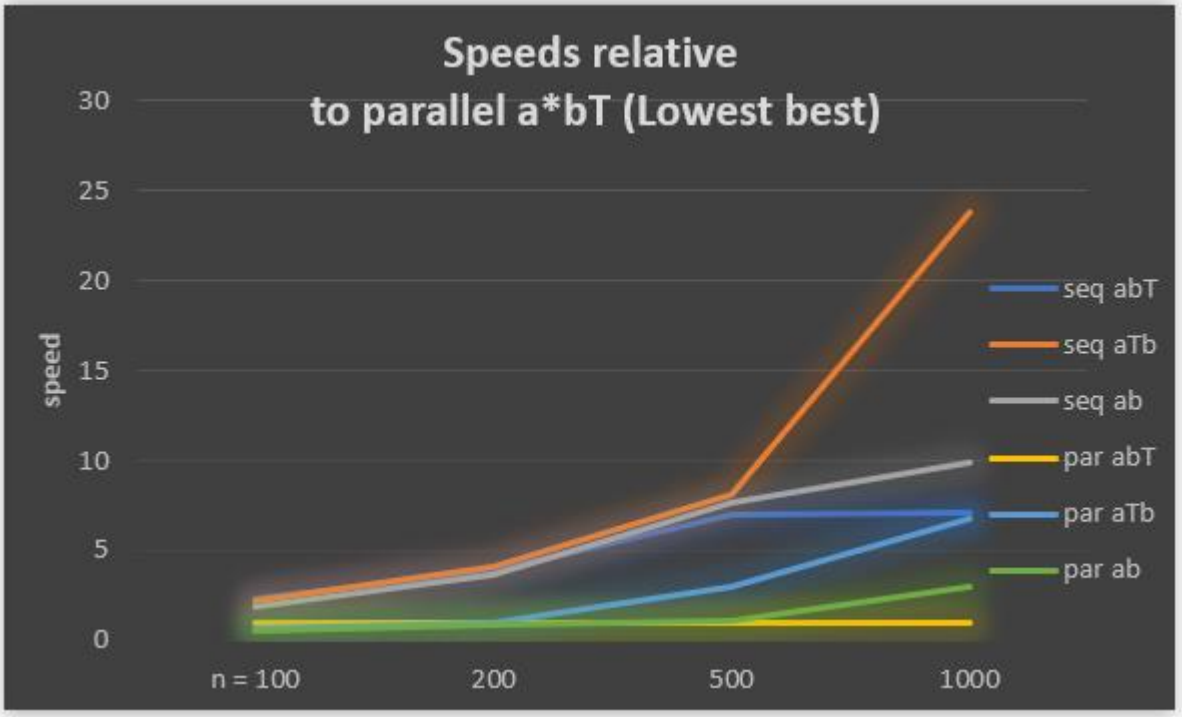
$$c[i][j] = \sum_{j=0}^{n}\sum_{k=0}^{n} aT[i][k] * b[j][k]$$

Will prove to be solved faster than (a*b) even though it has also $n^3+n^{2.}$operatoins, given that n is sufficiently large

All of these solutions have also been solved parallel. The way they have been done that is that each thread get assigned a portion of the 2 matrices, and put into a shared matrix called c. there is no need for synchronising since they don't access the same indexes in the c matrix. The transponation method is also implemented to be able to be parallel.

n = 100	seq	seq solves/parabt	par	par solves/par abt	speedup
abt	9,870621	2,313854498	4,265878	1	2,3138545
atb	9,635164	2,258659062	2,768199	0,648916589	3,48066161
ab	7,919805	1,856547468	2,415013	0,566123316	3,27940471
200					
	49,47439	3,920115189	12,620647	1	3,92011519
	51,803674	4,104676567	12,934721	1,024885729	4,005009
	46,544216	3,687942147	10,477041	0,830150863	4,44249631
500					
	694,69262	6,949806339	99,958558	1	6,94980634
	808,940172	8,09275552	295,139277	2,952616393	2,74087604
	766,951809	7,672697809	105,076189	1,051197527	7,29900671
1000					
	5850,42839	7,051325687	829,691982	1	7,05132569
	19789,6371	23,85178775	5599,586608	6,748994482	3,53412465
	8174,00935	9,851860124	2494,566121	3,006617124	3,27672587





From the data we get we can see a transposed * b is a horrible solution. We can also see a speedup if we use the parallel version compared to sequential even at n = 100. But still n = 100 would mean at least 100*100*100 = 1000000 operations. We can also see that until n = 500 there is very little use a*b transposed instead of a*b

These numbers were generated by running the program on a computer with a Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, 2601 Mhz, 4 cores, 8 logical processors

To run the program type “javac *java && java Oblig2 n seed” where n is the size the matrices will be having, and the seed is the seed of the matrices generated.

The conclusion is to use parallel a*b calculation until n = 500 and after use a*b transpose.