Memory efficient scheduling of Strassen-Winograd's matrix multiplication algorithm

Jean-Guillaume Dumas, Clément Pernet

Memory efficient scheduling of Strassen-Winograd's matrix multiplication algorithm

Jean-Guillaume Dumas
Laboratoire J. Kuntzmann
Université J. Fourier.
UMR CNRS 5224, BP 53X
F38041 Grenoble, France
jgdumas@imag.fr

Clément Pernet
School of Computer Science
University of Waterloo
Waterloo, ON,
N2B 3G1, Canada
cpernet@uwaterloo.ca

July 16, 2007

1 Introduction

Strassen's algorithm [8] was the first sub-cubic algorithm for matrix multiplication. Its improvement by Winograd led to a highly practicable algorithm. Former studies on this algorithm can be found in [4, 5, 1] and references therein for numerical computation and in [7, 2] for computations over a finite field. We do not consider here stability issues, just the number of arithmetic operations and memory allocations. Further studies have thus to be made in order to use these schedules in a numerical environment. They can nonetheless be used as is in an exact setting for instance for integer/rational or finite field applications [3].

In this report, we propose new schedules of the algorithm, that reduce of the extra memory allocation, by two differents means : either by introducing a few pre-additions, or by overwritting the input matrices.

2 Algorithm and notations

We first recall the principle of the algorithm, and setup the notations that will be used throughout the paper.

Let m, n and k be powers of 2. Let A and B be two matrices of dimension $m \times k$ and $k \times n$ and let $C = A \times B$.

Consider the natural block decomposition

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix},$$

where A_{11} and B_{11} have respectively dimension $m/2 \times k/2$ and $k/2 \times n/2$.

Winograd's algorithm computes the $m \times n$ matrix $C = A \times B$ with the following 22 block operations:

- 8 additions:

$$\begin{array}{ll} S_1 \leftarrow A_{21} + A_{22} & T_1 \leftarrow B_{12} - B_{11} \\ S_2 \leftarrow S_1 - A_{11} & T_2 \leftarrow B_{22} - T_1 \\ S_3 \leftarrow A_{11} - A_{21} & T_3 \leftarrow B_{22} - B_{12} \\ S_4 \leftarrow A_{12} - S_2 & T_4 \leftarrow T_2 - B_{21} \end{array}$$

- 7 recursive multiplications:

$$\begin{array}{ll} P_1 \leftarrow A_{11} \times B_{11} & P_5 \leftarrow S_1 \times T_1 \\ P_2 \leftarrow A_{12} \times B_{21} & P_6 \leftarrow S_2 \times T_2 \\ P_3 \leftarrow S_4 \times B_{22} & P_7 \leftarrow S_3 \times T_3 \\ P_4 \leftarrow A_{22} \times T_4 & \end{array}$$

- 7 final additions:

$$\begin{array}{ll} U_1 \leftarrow P_1 + P_2 & U_5 \leftarrow U_4 + P_3 \\ U_2 \leftarrow P_1 + P_6 & U_6 \leftarrow U_3 - P_4 \\ U_3 \leftarrow U_2 + P_7 & U_7 \leftarrow U_3 + P_5 \\ U_4 \leftarrow U_2 + P_5 & \end{array}$$

- The result is the matrix: $C = \begin{bmatrix} U1 & U5 \\ U6 & U7 \end{bmatrix}$

Figure 1 illustrates the dependencies between these tasks.

3 Memory efficient scheduling

Unlike the classic multiplication algorithm, Winograd's algorithm requires some extra temporary memory allocations to perform its 22 block operations. We present in this section several scheduling minimizing the number of temporary space allocated. Section 3.2 deals with the usual situation where the input matrices A and B are constant. In section 3.3, we allow to overwrite the input matrices A and B, leading to better memory efficiency.

3.1 Exhaustive search algorithm

We used a classical brute force search algorithm to get some of the new schedules that will be presented in the following sections. It has two components a `Tester` and an `Explorer`. The `Tester` is a variant of the pebble game of [5] with the following rules applied in this order:

Rule 1 A pebble is removed of any vertex having all its immediate succesors completed except for non-overwritable initial inputs.

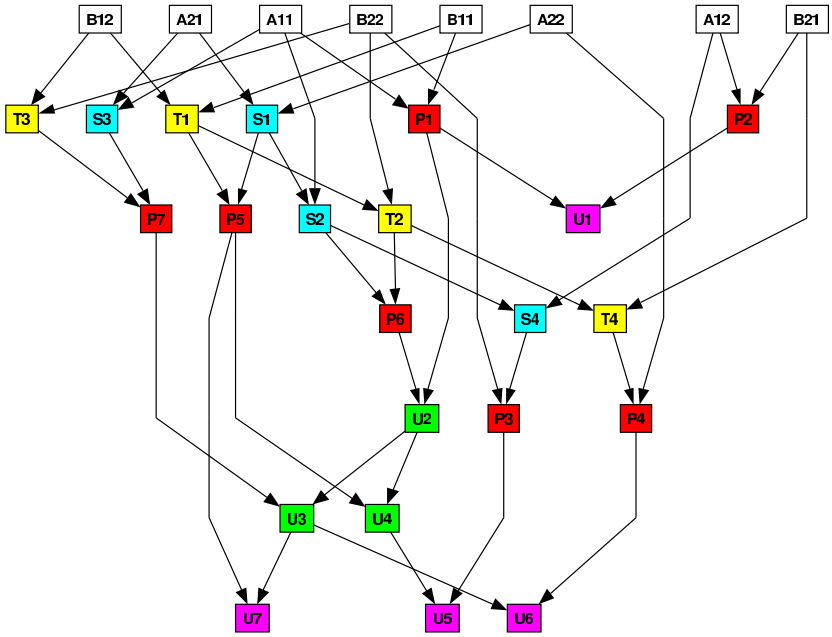

Figure 1: Task dependency graph of Winograd's algorithm

Rule 2 If all the immediate predecessors of a vertex have pebbles on them, if the computation to be performed at that vertex is of type $\alpha A \times B + \beta C$, and if all the other immediate succesors of C are already computed, then C's pebble can be moved onto that vertex.

Rule 3 If all the immediate predecessors of a vertex have pebbles on them, if the computation to be performed at that vertex is an addition, and if all the other immediate succesors of a predecessor are already computed, then this predecessor's pebble can be moved onto that vertex.

Rule 4 If all the immediate predecessors of a vertex are either initial inputs or have pebbles on them, a pebble may be placed on that vertex.

Then, the Explorer follows the dependency graph depth-first as in game theory: possible moves are ready (all the immediate predecessors are already computed) and not yet computed tasks ; the possible moves are tried recursively in turns.

3.2 With constant input matrices

3.2.1 Standard product

We first consider the basic operation $C \leftarrow A \times B$. The best known schedule for this case was given by [1]. We reproduce a similar schedule in table 1. It requires

#	operation	loc.	#	operation	loc.
1	$S_3 = A_{11} - A_{21}$	X_1	12	$P_1 = A_{11}B_{11}$	X_1
2	$T_3 = B_{22} - B_{12}$	X_2	13	$U_2 = P_1 + P_6$	C_{12}
3	$P_7 = S_3T_3$	C_{21}	14	$U_3 = U_2 + P_7$	C_{21}
4	$S_1 = A_{21} + A_{22}$	X_1	15	$U_4 = U_2 + P_5$	C_{12}
5	$T_1 = B_{12} - B_{11}$	X_2	16	$U_7 = U_3 + P_5$	C_{22}
6	$P_5 = S_1T_1$	C_{22}	17	$U_5 = U_4 + P_3$	C_{12}
7	$S_2 = S_1 - A_{11}$	X_1	18	$T_4 = T_2 - B_{21}$	X_2
8	$T_2 = B_{22} - T_1$	X_2	19	$P_4 = A_{22}T_4$	C_{11}
9	$P_6 = S_2T_2$	C_{12}	20	$U_6 = U_3 - P_4$	C_{21}
10	$S_4 = A_{12} - S_2$	X_1	21	$P_2 = A_{12}B_{21}$	C_{11}
11	$P_3 = S_4B_{22}$	C_{11}	22	$U_1 = P_1 + P_2$	C_{11}

Table 1: Winograd's algorithm for operation $C \leftarrow AB$, with two temporaries

two temporary blocks X_1 and X_2 of dimension respectively $m/2 \times \max(k/2, n/2)$ and $k/2 \times n/2$. Assuming $m = n = k$, and summing these temporary allocations for every recursive level, leads to a total extra memory requirement of

$$2\sum_{i=1}^{\log n} \left(\frac{n}{2^i}\right)^2 < \frac{2}{3}n^2.$$

3.2.2 Product with accumulation

For the more general operation $C \leftarrow \alpha A \times B + \beta C$, a first naive method would be to compute the product $\alpha A \times B$ using the scheduling of table 1, into a temporary matrix C' and finally compute $C \leftarrow C' + \beta C$. It would require $(1+2/3)n^2$ extra memory allocation.

Now the schedule of table 2 due to [5, fig. 6] only requires 3 temporary blocks for the same number of operations (7 multiplications and $4 + 15$ additions).

The three temporary blocks X_1, X_2, X_3 required have dimension $m/2 \times n/2$, $m/2 \times k/2$ and $k/2 \times n/2$. Assuming $m = n = k$, and summing these temporary allocations for every recursive level, leads to a total extra memory requirement of

$$3\sum_{i=1}^{\log n} \left(\frac{n}{2^i}\right)^2 < n^2.$$

We propose in table 3 a new schedule for the same operation $\alpha A \times B + \beta C$ only requiring two temporary blocks. The price to pay for this improvement is three pre-additions on the input matrix C. Again, the two temporary blocks X_1 and X_2 of dimension respectively $m/2 \times \max(k/2, n/2)$ and $k/2 \times n/2$. Assuming $m = n = k$, and summing these temporary allocations for every recursive level,

#	operation	loc.	#	operation	loc.
1	$S_1 = A_{21} + A_{22}$	X_1	12	$S_4 = A_{12} - S_2$	X_1
2	$T_1 = B_{12} - B_{11}$	X_2	13	$T_4 = T_2 - B_{21}$	X_2
3	$P_5 = \alpha S_1 T_1$	X_3	14	$C_{12} = \alpha S_4 B_{22} + C_{12}$	C_{12}
4	$C_{22} = P_5 + \beta C_{22}$	C_{22}	15	$U_5 = U_2 + C_{12}$	C_{12}
5	$C_{12} = P_5 + \beta C_{12}$	C_{12}	16	$P_4 = \alpha A_{12} T_4 - \beta C_{21}$	C_{21}
6	$S_2 = S_1 - A_{11}$	X_1	17	$S_3 = A_{11} - A_{21}$	X_1
7	$T_2 = B_{22} - T_1$	X_2	18	$T_3 = B_{22} - B_{12}$	X_2
8	$P_1 = \alpha A_{11} B_{11}$	X_3	19	$U_3 = \alpha S_3 T_3 + U_2$	X_3
9	$C_{11} = P_1 + \beta C_{11}$	C_{11}	20	$U_7 = U_3 + C_{22}$	C_{22}
10	$U_2 = \alpha S_2 T_2 + P_1$	X_3	21	$U_6 = U_3 - C_{21}$	C_{21}
11	$U_1 = \alpha A_{12} B_{21} + C_{11}$	C_{11}	22		

Table 2: Schedule for operation $C \leftarrow AB + \beta C$ with 3 temporaries

#	operation	loc.	#	operation	loc.
1	$C_{22} = C_{22} - C_{12}$	C_{22}	13	$P_3 = \alpha S_4 B_{22} + C_{12}$	C_{12}
2	$C_{21} = C_{21} - C_{22}$	C_{21}	14	$P_1 = \alpha A_{11} B_{11}$	X_1
3	$C_{12} = C_{12} - C_{22}$	C_{12}	15	$U_2 = P_6 + P1$	C_{21}
4	$S_1 = A_{21} + A_{22}$	X_1	16	$P_2 = \alpha A_{12} B_{21} + \beta C_{11}$	C_{11}
5	$T_1 = B_{12} - B_{11}$	X_2	17	$U_1 = P_1 + P_2$	C_{11}
6	$P_5 = \alpha S_1 T_1 + \beta C_{12}$	C_{12}	18	$U_5 = U_2 + C_{12}$	C_{12}
7	$S_2 = S_1 - A_{11}$	X_1	19	$S_3 = A_{11} - A_{21}$	X_1
8	$T_2 = B_{22} - T_1$	X_2	20	$T_3 = B_{22} - B_{12}$	X_2
9	$P_6 = \alpha S_2 T_2 + \beta C_{21}$	C_{21}	21	$U_3 = P_7 + U_2 = \alpha S_3 T_3 + U_2$	C_{21}
10	$S_4 = A_{12} - S_2$	X_1	22	$U_7 = U_3 + C_{22}$	C_{22}
11	$T_4 = T_2 - B_{21}$	X_2	23	$U_6 = U_3 - P_4 = -\alpha A_{12} T_4 + U_3$	C_{21}
12	$C_{22} = P_5 + \beta C_{22}$	C_{22}			

Table 3: Schedule for operation $C \leftarrow AB + \beta C$ with 2 temporaries

leads to a total extra memory requirement of

$$2\sum_{i=1}^{\log n}\left(\frac{n}{2^i}\right)^2 < \frac{2}{3}n^2.$$

3.3 Overwriting the input matrices

We now relax some constraints on the previous problem: the input matrices A and B can be overwritten. Although such an assumption is not feasible for the design of a general matrix multiplication routine, some specific application can allow it: for instance the computation of the rank, using a block elimination algorithm, would enable this.

3.3.1 Standard product

We propose in table 4 a new schedule that computes the product $C \leftarrow A \times B$ without any temporary memory allocation. The point here is to find an ordering where the recursive calls can be made also in place (i.e. such that the operand of a multiplication are no longer in use after the multiplication). The exhaustive search showed that no schedule exists overwriting less than four sub-blocks.

#	operation	loc.	#	operation	loc.
1	$S_3 = A_{11} - A_{21}$	C_{11}	12	$S_4 = A_{12} - S_2$	C_{22}
2	$S_1 = A_{21} + A_{22}$	A_{21}	13	$P_6 = S_2T_2$	C_{12}
3	$T_1 = B_{12} - B_{11}$	C_{22}	14	$U_2 = P_1 + P_6$	C_{12}
4	$T_3 = B_{22} - B_{12}$	B_{12}	15	$U_3 = U_2 + P_7$	C_{21}
5	$P_7 = S_3T_3$	C_{21}	16	$P_3 = S_4B_{22}$	B_{11}
6	$S_2 = S_1 - A_{11}$	B_{12}	17	$U_7 = U_3 + P_5$	C_{22}
7	$P_1 = A_{11}B_{11}$	C_{11}	18	$U_6 = U_3 - P_4$	C_{21}
8	$T_2 = B_{22} - T_1$	B_{11}	19	$U_4 = U_2 + P_5$	C_{12}
9	$P_5 = S_1T_1$	A_{11}	20	$U_5 = U_4 + P_3$	C_{12}
10	$T_4 = T_2 - B_{21}$	C_{22}	21	$P_2 = A_{12}B_{21}$	B_{11}
11	$P_4 = A_{22}T_4$	A_{21}	22	$U_1 = P_1 + P_2$	C_{11}

Table 4: Schedule for operation $C \leftarrow AB$ in place

Note that this schedule uses only two blocks of A or B as extra temporaries (namely A_{11}, A_{21}, B_{11} and B_{12}) but overwrites the whole of A and B. For instance the recursive computation of P_2 requires to also overwrite parts of A_{12} and B_{21}. This schedule can nonetheless overwrite strictly only two blocks of A and two blocks of B. This is achieved by making a backup of the overwriten parts into some available extra memory. The schedule is then modified as follows:

#	operation	loc.
10bis	`Copy` A_{22} into C_{12}	C_{12}
11	$P_4 = A_{22}T_4$	A_{21}
11bis	`Restore` A_{22} from C_{12}	A_{22}
15bis	`Copy` B_{22} into B_{12}	B_{12}
16	$P_3 = S_4B_{22}$	B_{11}
16bis	`Restore` B_{22} from B_{12}	B_{22}
20bis	`Copy` A_{12} into A_{21}	A_{21}
20ter	`Copy` B_{21} into B_{12}	B_{12}
21	$P_2 = A_{12}B_{21}$	B_{11}
21bis	`Restore` B_{21} from B_{12}	B_{21}
21ter	`Restore` A_{12} from A_{21}	A_{12}

In the following, we will denote by `IP` for `I`nPlace, either one of theses two schedules.

We thus also present in tables 5 and 6 two new schedules overwriting only one of the two input matrices, but requiring an extra temporary space. These two schedules are denoted `IPLeft` and `IPRight`. Here also, the exhaustive search showed that no schedule exists overwriting only one side and not using extra temporary.

#	operation	loc.	#	operation	loc.
1	$S_3 = A_{11} - A_{21}$	C_{22}	13	$T_4 = T_2 - B_{21}$	A_{11}
2	$S_1 = A_{21} + A_{22}$	A_{21}	14	$U_2 = P_1 + P_6$	C_{21}
3	$S_2 = S_1 - A_{11}$	C_{12}	15	$U_4 = U_2 + P_5$	C_{12}
4	$T_1 = B_{12} - B_{11}$	C_{21}	16	$U_3 = U_2 + P_7$	C_{21}
5	$P_1 = \texttt{IPLeft}(A_{11}B_{11})$	C_{11}	17	$U_7 = U_3 + P_5$	C_{22}
6	$T_3 = B_{22} - B_{12}$	A_{11}	18	$U_5 = U_4 + P_3$	C_{12}
7	$P_7 = \texttt{IP}(S_3T_3)$	X_1		$A_{21} = \texttt{Copy}(A_{12})$	A_{21}
8	$T_2 = B_{22} - T_1$	A_{11}	19	$P_2 = \texttt{IPLeft}(A_{12}B_{21})$	X_1
9	$P_5 = \texttt{IP}(S_1T_1)$	C_{22}		$A_{12} = \texttt{Copy}(A_{21})$	A_{12}
10	$S_4 = A_{12} - S_2$	C_{21}	20	$U_1 = P_1 + P_2$	C_{11}
11	$P_3 = \texttt{IPLeft}(S_4B_{22})$	A_{21}	21	$P_4 = \texttt{IPRight}(A_{22}T_4)$	A_{21}
12	$P_6 = \texttt{IPLeft}(S_2T_2)$	C_{21}	22	$U_6 = U_3 - P_4$	C_{21}

Table 5: `IPLeft` Schedule for operation $C \leftarrow AB$ using two blocks of A and one temporary

The exhaustive search showed that no schedule exists overwriting only one block of A. Remark that if it is allowed to use three blocks of A, the two `Copy` operations of table 5 can be avoided. Note also that if three blocks of A can be overwriten then the last multiplication ($P4$) can also be made by a strict recursive call to `IPLeft`. Both behaviors can be simultaneous if four blocks of A (the whole matrix) is overwritable.

Here also the copy or the call to `IPLeft` for P_3 can be avoided if three or four blocks of B are overwritable.

#	operation	loc.	#	operation	loc.
1	$S_3 = A_{11} - A_{21}$	C_{22}	13	$S_4 = A_{12} - S_2$	B_{11}
2	$S_1 = A_{21} + A_{22}$	C_{21}	14	$U_2 = P_1 + P_6$	C_{21}
3	$T_1 = B_{12} - B_{11}$	C_{12}	15	$U_4 = U_2 + P_5$	C_{12}
4	$P_1 = \texttt{IPRight}(A_{11}B_{11})$	C_{11}	16	$U_3 = U_2 + P_7$	C_{21}
5	$S_2 = S_1 - A_{11}$	B_{11}	17	$U_7 = U_3 + P_5$	C_{22}
6	$T_3 = B_{22} - B_{12}$	B_{12}	18	$U_6 = U_3 - P_4$	C_{21}
7	$P_7 = \texttt{IP}(S_3T_3)$	X_1	19	$P_3 = \texttt{IPLeft}(S_4B_{22})$	B_{12}
8	$T_2 = B_{22} - T_1$	B_{12}	20	$U_5 = U_4 + P_3$	C_{12}
9	$P_5 = \texttt{IP}(S_1T_1)$	C_{22}		$B_{11} = \texttt{Copy}(B_{21})$	B_{11}
10	$T_4 = T_2 - B_{21}$	C_{12}	21	$P_2 = \texttt{IPRight}(A_{12}B_{21})$	B_{12}
11	$P_6 = \texttt{IPRight}(S_2T_2)$	C_{21}		$B_{21} = \texttt{Copy}(B_{11})$	B_{21}
12	$P_4 = \texttt{IPRight}(A_{22}T_4)$	B_{12}	22	$U_1 = P_1 + P_2$	C_{11}

Table 6: `IPRight` Schedule for operation $C \leftarrow AB$ using two block of B and one temporary

3.3.2 Product with accumulation

We now consider the general operation $C \leftarrow \alpha A \times B + \beta C$, where the input matrices A and B can be overwritten. We propose in table 7 a schedule that only requires 2 temporary block matrices, instead of the 3 of table 2. Again, this is achieved at the price of two additional pre-additions on the matrix C. Compared to the scheduling of table 3, the possibility to overwrite the input matrices makes it possible to save one pre-addition.

#	operation	loc.	#	operation	loc.
1	$C_{21} = C_{21} - C_{22}$	C_{21}	13	$P_4 = A_{22}T_4 + \beta C_{21}$	C_{21}
2	$C_{22} = C_{22} - C_{12}$	C_{22}	14	$P_2 = A_{12}B_{21} + \beta C_{11}$	C_{11}
3	$S_3 = A_{11} - A_{21}$	X	15	$P_1 = \texttt{IP}(A_{11}B_{11})$	B_{21}
4	$T_3 = B_{22} - B_{12}$	Y	16	$U_1 = P_1 + P_2$	C_{11}
5	$P_7 = S_3T_3 + \beta C_{22}$	C_{22}	17	$P_6 = \texttt{IP}(S_2T_2)$	A_{12}
6	$S_1 = A_{21} + A_{22}$	A_{21}	18	$U_2 = P_1 + P_6$	C_{12}
7	$T_1 = B_{12} - B_{11}$	B_{12}	19	$U_4 = U_2 + P_5$	C_{12}
8	$S_2 = S_1 - A_{11}$	X	20	$U_3 = U_2 + P_7$	C_{22}
9	$T_2 = B_{22} - T_1$	Y	21	$U_7 = U_3 + P_5$	C_{22}
10	$P_5 = S_1T_1 + \beta C_{12}$	C_{12}	22	$U_6 = U_3 - P_4$	C_{21}
11	$S_4 = A_{12} - S_2$	A_{21}	23	$P_3 = \texttt{IP}(S_4B_{22})$	A_{12}
12	$T_4 = T_2 - B_{21}$	B_{12}	24	$U_5 = U_4 + P_3$	C_{12}

Table 7: Schedule for $C \leftarrow \alpha AB + \beta C$ with 2 temporaries and overwriting A and B

4 Conclusion

With constant input matrices, we reduced the number of extra memory allocations for the operation $C \leftarrow \alpha A \times B + \beta C$ from n^2 to $\frac{2}{3}n^2$, by introducing three extra pre-additions. As shown below the overhead induced by these supplementary additions is fully covered by the gains in number of memory allocations.

If the input matrices can be overwritten, we proposed a fully *in-place* schedule for the operation $C \leftarrow A \times B$ without any extra operation. We also reduced the extra memory allocations for the operation $C \leftarrow \alpha A \times B + \beta C$ from n^2 to $\frac{2}{3}n^2$, by introducing only two extra pre-additions. Finally, we also proposed variants for the operation $C \leftarrow A \times B$, where only one of the input matrices is being overwritten and one temporary is required. These schedules can prove useful e.g. when computing in place LU decomposition (see e.g. [6] for the in place exact LQUP decomposition) where some intermediate results can be overwritten.

Table 8 gives a summary of the features of each schedule that has been presented. The complexities are given only for $m = k = n$ being a power of 2.

Theorem 1. *The arithmetic and memory complexities given in table 8 are correct.*

Proof. For $A \times B$, the arithmetic complexity satisfies classically

$$\left\{ \begin{array}{rcl} W(n) & = & 7W(\frac{n}{2}) + 15n^2 \\ W(1) & = & 1 \end{array} \right. ,$$

so that $W(n) = 6n^{\log_2(7)} - 5n^2$.

The schedule of [1] requires

$$\left\{ \begin{array}{rcl} M_1(n) & = & 2\left(\frac{n}{2}\right)^2 + M_1\left(\frac{n}{2}\right) \\ M_1(1) & = & 0 \end{array} \right.$$

extra memory space, which is $M_1(n) = \frac{2}{3}n^2$. Its total number of allocations satifies

$$T_1(n) = 2\left(\frac{n}{2}\right)^2 + 7T_1\left(\frac{n}{2}\right)$$

which is $T_1(n) = \frac{2}{3}(n^{\log_2(7)} - n^2)$.

The schedule of Table 5 requires

$$M_5(n) = \left(\frac{n}{2}\right)^2 + M_5\left(\frac{n}{2}\right)$$

extra memory space, which is $M_5(n) = \frac{1}{3}n^2$. Its total number of allocations satifies

$$T_5(n) = \left(\frac{n}{2}\right)^2 + 5T_5\left(\frac{n}{2}\right)$$

which is $T_5(n) = n^{\log_2(5)} - n^2$.

	Algorithm	Input matrices	# of extra temporaries	total extra memory	total # of allocations	arithmetic complexity
$A \times B$	[1]	Constant	2	$\frac{2}{3}n^2$	$\frac{2}{3}(n^{2.807} - n^2)$	$6n^{2.807} - 5n^2$
	Table 4	Both Overwritten	0	0	0	$6n^{2.807} - 5n^2$
	Table 5	A Overwritten	1	$\frac{1}{3}n^2$	$n^{2.322} - n^2$	$6n^{2.807} - 5n^2$
	Table 6	B Overwritten	1	$\frac{1}{3}n^2$	$n^{2.322} - n^2$	$6n^{2.807} - 5n^2$
$\alpha A \times B + \beta C$	[5]	Constant	3	n^2	$\frac{1}{3}\left(14n^{2.807} + 7n^2 - 21n^{2.322}\right)$	$6n^{2.807} - 4n^2$
	Table 3	Constant	2	$\frac{2}{3}n^2$	$\frac{1}{3}\left(14n^{2.807} + 7n^2 - 15n^{2.585}\right)$	$6n^{2.807} - 4n^2 + \frac{3}{2}(n^{2.585} - n^2)$
	Table 7	Overwritten	2	$\frac{2}{3}n^2$	$\frac{1}{2}n^2 \log_2(n)$	$6n^{2.807} - 4n^2 + \frac{1}{2}n^2 \log_2(n)$

Table 8: Summary of the schedules presented for Winograd's algorithm

The schedule of Table 6 requires the same amount of arithmetic operations or memory.

For $A \times B + \beta C$, the arithmetic complexity of [5] satifies

$$W_2(n) = 5W_2\left(\frac{n}{2}\right) + 2W_1\left(\frac{n}{2}\right) + 14\left(\frac{n}{2}\right)^2,$$

so that $W_2(n) = 6n^{\log_2(7)} - 4n^2$; its number of extra memory satifies

$$M_2(n) = 3\left(\frac{n}{2}\right)^2 + M_2\left(\frac{n}{2}\right),$$

which is $M_2(n) = n^2$; its total number of allocations satifies

$$T_2(n) = 3\left(\frac{n}{2}\right)^2 + 5T_2\left(\frac{n}{2}\right) + 2T_1(n),$$

which is $T_2(n) = \frac{1}{3}\left(14n^{\log_2(7)} + 7n^2 - 21n^{\log_2(5)}\right)$.

The arithmetic complexity of algorithm 3 satifies

$$W_3(n) = 6W_3\left(\frac{n}{2}\right) + W_1\left(\frac{n}{2}\right) + 16\left(\frac{n}{2}\right)^2,$$

so that $W_3(n) = 6n^{\log_2(7)} - 4n^2 + \frac{3}{2}(n^{\log_2(6)} - n^2)$; its number of extra memory satifies

$$M_3(n) = 2\left(\frac{n}{2}\right)^2 + M_3\left(\frac{n}{2}\right),$$

which is $M_3(n) = \frac{2}{3}n^2$; its total number of allocations satifies

$$T_3(n) = 2\left(\frac{n}{2}\right)^2 + 6T_3\left(\frac{n}{2}\right) + T_1(n),$$

which is $T_3(n) = \frac{1}{3}\left(14n^{\log_2(7)} + 7n^2 - 15n^{\log_2(6)}\right)$.

The arithmetic complexity of algorithm 7 satifies

$$W_7(n) = 4W_7\left(\frac{n}{2}\right) + 3W_1\left(\frac{n}{2}\right) + 17\left(\frac{n}{2}\right)^2,$$

so that $W_7(n) = 6n^{\log_2(7)} - 4n^2 + \frac{1}{2}n^2\log_2(n)$; its number of extra memory satifies

$$M_7(n) = 2\left(\frac{n}{2}\right)^2 + M_7\left(\frac{n}{2}\right),$$

which is $M_7(n) = \frac{2}{3}n^2$; its total number of allocations satifies

$$T_7(n) = 2\left(\frac{n}{2}\right)^2 + 4T_7\left(\frac{n}{2}\right),$$

which is $T_7(n) = \frac{1}{2}n^2\log_2(n)$. □

Four instance, if we sum up allocations and arithmetic operations, we see that our schedule of table 3 makes less operations than that of [5] as soon as $n \geq 10$.

References

[1] C. C. Douglas, M. Heroux, G. Slishman, and R. M. Smith. GEMMW: A portable level 3 BLAS Winograd variant of Strassen's matrix-matrix multiply algorithm. *Journal of Computational Physics*, 110:1–10, 1994.

[2] J.-G. Dumas, T. Gautier, and C. Pernet. Finite field linear algebra subroutines. In T. Mora, editor, *ISSAC'2002*, pages 63–74. ACM Press, New York, July 2002.

[3] J.-G. Dumas, P. Giorgi, and C. Pernet. FFPACK: Finite field linear algebra package. In J. Gutierrez, editor, *ISSAC'2004*, pages 119–126. ACM Press, New York, July 2004.

[4] S. Huss-Lederman, E. M. Jacobson, J. R. Johnson, A. Tsao, and T. Turnbull. Implementation of Strassen's algorithm for matrix multiplication. In ACM, editor, *Supercomputing '96 Conference Proceedings: November 17–22, Pittsburgh, PA*. ACM Press and IEEE Computer Society Press, 1996. `www.supercomp.org/sc96/proceedings/SC96PROC/JACOBSON/`.

[5] S. Huss-Lederman, E. M. Jacobson, J. R. Johnson, A. Tsao, and T. Turnbull. Strassen's algorithm for matrix multiplication : Modeling analysis, and implementation. Technical report, Center for Computing Sciences, Nov. 1996. CCS-TR-96-17.

[6] O. H. Ibarra, S. Moran, and R. Hui. A generalization of the fast LUP matrix decomposition algorithm and applications. *Journal of Algorithms*, 3(1):45–56, Mar. 1982.

[7] C. Pernet. Implementation of Winograd's fast matrix multiplication over finite fields using ATLAS level 3 BLAS. Technical report, Laboratoire Informatique et Distribution, July 2001. `www-id.imag.fr/Apache/RR/RR011122FFLAS.ps.gz`.

[8] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.