

APPENDIX B

TESTING THE SPEED AND MEMORY USAGE OF MULTIPROD'S ENGINES

Testing speed

The M-function “testing_ARRAYLAB_engines.m” was used to test the speed of different engines to perform multiple matrix products. Several new engines were compared to the engine used in MULTIPROD 1.3. Some of the engines were similar to the engine used in MULTIPROD 1.3 (ARRAYLAB 1.3, ARRAYLAB 1.31, ARRAYLAB 1.32, ARRAYLAB 1.32), and were not capable of performing AX. Plain loops were also tested.

The new engines are listed in the tables below. All of them happen to be based on some kind of “reshape technique”. Most use some form of singleton expansion (SX), even when the external dimensions of A and B are identical. All of them eventually feed the reshaped and/or expanded arrays either to TIMES or MTIMES. In short, the tested engines use two kinds of pipelines:

1. (PERMUTE) → 4-D RESHAPE → SX → TIMES → SUM → 3-D RESHAPE → (PERMUTE)
2. PERMUTE → 2-D RESHAPE → MTIMES → 3-D RESHAPE → PERMUTE

Notice that, in pipeline 1, 4-D RESHAPE expands a 3-D array to 4-D, by adding a singleton dimension, while in pipeline 2, 2-D RESHAPE rearranges or “squashes” a 3-D array to 2-D. Moreover, in pipeline 1 sometimes the initial permutation is not needed, and SX can be performed in four different ways. Namely, by means of Tony’s trick, or functions GENOP, BSX_TIMES, or BSXFUN. Tony’s trick performs SX physically (by allocating new memory to store the replicated array), while GENOP, BSX_TIMES and BSXFUN are preferable because they perform it virtually (without allocating new memory).

The main results of the speed tests for some of these engines are summarized in figure 1. Normalized processing times (ratio between processing time and average processing time) were reported to facilitate comparison between two systems, a PC running MATLAB R2007b, and a PC running MATLAB R2008b.

Table 1. Engines used to solve a multiproduct which did neither require SX nor AX to match the external dimensions of A and B. A is $N \times (P \times Q)$, B is $N \times (Q \times R)$.

Pipeline	Engine	RESHAPE technique	Multiplication method
(PERMUTE) 4D-RESH SX TIMES SUM	resh4D_clone (Tony’s trick)		A = A(:, :, :, ones(1, R)) B = B(:, :, :, ones(1, P)) C = A .* B
	resh4D_genop	A = reshape(A, [N P Q 1]) B = reshape(B, [N 1 Q R])	C = genop(@times, A, B)
	resh4D_bsxtimes		C = bsx_times(A, B)
	resh4D_bsxfun		C = bsxfun(@times, A, B)
	permA_resh4D_ bsxfun	A = reshape(A, [N Q P 1]) B = reshape(B, [N Q 1 R])	C = bsxfun(@times, A, B)
	permAB_resh4D_ bsxfun	A = reshape(A, [Q P 1 N]) B = reshape(B, [Q 1 R N])	C = bsxfun(@times, A, B)

Table 2. Engines used to solve a multiproduct which required AX. Namely, the multiproduct of a single-block array by an array with N blocks, or vice versa. Either A is (P×Q) and B is N×(Q×R), or A is N×(P×Q) and B is (Q×R).

Pipeline	Engine	RESHAPE technique	Multiplication method
4D-RESH SX TIMES SUM	resh4D_bsxfun_expA	A = reshape(A, [1 P Q 1]) B = reshape(B, [N 1 Q R])	C = bsxfun(@times,A,B)
	resh4D_bsxfun_expB	A = reshape(A, [N P Q 1]) B = reshape(B, [1 1 Q R])	C = bsxfun(@times,A,B)
PERMUTE 2D-RESH MTIMES	squashB_mtimes	A is (P×Q) B = reshape(B, [Q, R*N])	C = A * B % mtimes
	squashA_mtimes	A = reshape(A, [N*P, Q]) B is (Q×R)	C = A * B % mtimes

As expected, conventional loops were markedly slower than the other engines (Loop 2 was the fastest “conventional loop” technique). The ARRAYLAB engines, similar to the engine used in MULTIPROD 1.3, were faster than “R4D Tony’s trick”, “R4D GENOP”, “R4D BSX_TIMES”, but slower than “R4D BSXFUN”. As for the matrix expansion experiments, the “SQUASH MTIMES” techniques were much faster than “R4D BSXFUN”, even though the latter exploits the builtin function BSXFUN. This is due to the fact that MTIMES calls a few Basic Linear Algebra Subroutines (BLAS) to perform the multiplication. The BLAS are CPU-specific interfaces provided by hardware vendors (such as Intel and AMD) to speed up computations.

MULTIPROD 2.0 implements generalizations of the “R4D BSXFUN” and “SQUASH MTIMES” engines, which are capable of operating on arrays of any size.

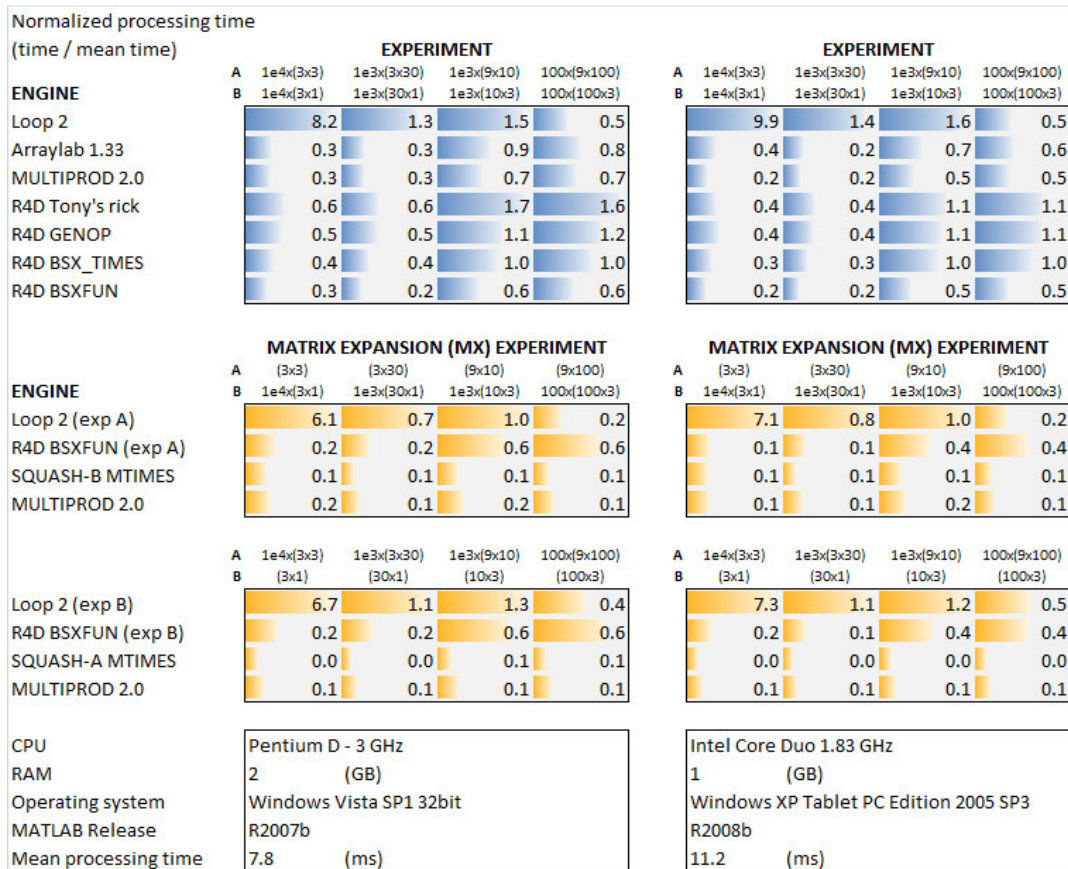


Figure 1. Notice that in the MX experiments, all the engines performed virtual expansion. The “SQUASH MTIMES” engines actually squashed the 3-D array to 2-D, rather than expanding the matrix.

Generalization of the R4D-BSXFUN engine (pipeline 1)

“R4D BSXFUN” was one of the six implementations of pipeline 1:

(PERMUTE) → 4-D RESHAPE → SX → TIMES → SUM → 3-D RESHAPE → (PERMUTE)

Since PERMUTE was not used by “R4D BSXFUN” and the SX and TIMES steps were both performed by BSXFUN, we can write:

4-D RESHAPE → BSXFUN → SUM → 3-D RESHAPE

A and B are reshaped from 3-D to 4-D by adding a singleton dimension, and C is reshaped back to 3-D by deleting a singleton dimension. Thus, we can write:

ADDSINGLETON → BSXFUN → SUM → DELSINGLETON

The same method can be used to process N-D arrays ($N > 2$). This pipeline was adapted to deal with different MULTIPROD syntaxes (see Appendix A):

SWAPPING → BSXFUN → SUM
ADDSINGLETON → BSXFUN
BSXFUN

In some simple cases, BSXFUN (i.e. SX → TIMES) was not needed, and TIMES was sufficient:

SWAPPING → TIMES → SUM
DELSINGLETON → TIMES → SUM
ADDSINGLETON → TIMES
TIMES → SUM
TIMES

Generalization of the SQUASH-MTIMES engines (pipeline 2)

Pipeline 2 was specifically designed to solve the matrix expansion problem. Pipeline 1 was capable of solving that problem as well, but was shown to be markedly slower in this specific task (Fig. 1). The two “SQUASH MTIMES” engines represented the only tested implementations of pipeline 2:

PERMUTE → 2-D RESHAPE → MTIMES → 3-D RESHAPE → PERMUTE

We can use the expressions 2-D SQUASH and 3-D REASSEMBLE to indicate the first two and last two steps of pipeline 2:

2-D SQUASH → MTIMES → 3-D REASSEMBLE

This pipeline was generalized to N-D and adapted to deal with different MULTIPROD syntaxes:

2-D SQUASH → MTIMES → N-D REASSEMBLE
RESHAPE → 2-D SQUASH → MTIMES → N-D REASSEMBLE → RESHAPE
RESHAPE → 2-D SQUASH → MTIMES → N-D REASSEMBLE

Herein RESHAPE means inserting or removing singletons. The initial RESHAPE is used to shift the single block array to dimension(s) [1 2] (e.g. when its size is $1 \times 1 \times (P \times Q)$), or 1, or 2 (e.g. when its size is $1 \times 1 \times (P)$) and/or to turn into 2-D blocks the 1-D blocks contained in the multi-block array (e.g. when its size is $M \times N \times (P)$). The final RESHAPE adjusts the output size, according to the rules of MULTIPROD.

A simplified version of this generalized pipeline was used to multiply two single-block arrays. This does not require array expansion:

RESHAPE → MTIMES → RESHAPE
RESHAPE → MTIMES
MTIMES

Testing memory usage

Function `testing_memory_usage.m` was used to test the memory efficiency of some of the above mentioned techniques for singleton expansion, as well as some MATLAB commands used to rearrange array elements (TRANSPOSE, RESHAPE, and PERMUTE). The results are shown in Figure 2. GENOP and the embedded Tony's trick needed to temporarily allocate a significant amount of additional memory to store the expanded array. On the other hand, BSXFUN and its replacement BSX_TIMES obtained the same result without additional memory allocation. These techniques for SX were also shown to be faster than GENOP and Tony's trick. Their memory efficient behavior is referred to as virtual expansion, as opposed to the physical expansion performed by GENOP and Tony's trick. BSXFUN was selected as the core of one of MULTIPROD's engines. BSX_TIMES is a replacement for BSXFUN which can be exploited by users of pre-R2007a releases of MATLAB, in which BSXFUN was not provided (see MULTIPROD manual).

PC: Lenovo Thinkpad X60 Tablet					
OS: Windows XP Tablet PC Edition 2005 SP3					
CPU: Intel Core Duo 1.83 GHz					
RAM: 1 GB					
MATLAB: R2008b (7.7)					
		Memory usage		Peak memory usage	
		Value	Increase	Value	Increase
		(MB)	(MB)	(MB)	(MB)
	Initial value	153.0		Idem	
	A1D = RAND(1, 1e6)	160.8	(+7.8)	Idem	
	A2D = RAND(100, 1e4)	168.6	(+7.8)	Idem	
Array rearrangement	TRANSPOSE(A1D)	168.6	(+0.0)	Idem	
	TRANSPOSE(A2D)	176.5	(+7.9)	Idem	
	RESHAPE(A2D, ...)	176.5	(+0.0)	Idem	
	PERMUTE(A2D, ...)	184.3	(+7.8)	Idem	
SX	Tony's trick	223.5	(+39.2)	Idem	
SX+ TIMES	BSXFUN	262.6	(+39.1)	Idem	
	BSX_TIMES	301.7	(+39.1)	Idem	
	GENOP	340.8	(+39.1)	364.2	(+62.5)
	Embedded T. trick	380.0	(+39.2)	419.1	(+78.3)

Figure 2. Memory used by MATLAB commands performing array element rearrangement and by functions implementing singleton expansion (SX), or singleton expansion associated with a multiplication (SX + TIMES). In this test, SX consists of a replication by 5 times of A2D. See function `testing_memory_usage.m` for further details.