Generalizing Matrix Multiplication for Efficient Computations on Modern Computers

Stanislav G. Sedukhin[1] and Marcin Paprzycki[2]

[1] The University of Aizu, Aizuwakamatsu City, Fukushima 965-8580, Japan
sedukhin@u-aizu.ac.jp
[2] Systems Research Institute Polish Academy of Sciences, Warsaw, Poland
marcin.paprzycki@ibspan.waw.pl

Abstract. Recent advances in computing allow taking new look at matrix multiplication, where the key ideas are: decreasing interest in recursion, development of processors with thousands (potentially millions) of processing units, and influences from the Algebraic Path Problems. In this context, we propose a generalized matrix-matrix multiply-add (MMA) operation and illustrate its usability. Furthermore, we elaborate the interrelation between this generalization and the BLAS standard.

Keywords: matrix matrix multiplication, algebraic semiring, path problem, fused multiply add, BLAS, matrix data manipulation.

1 Introduction

Dense matrix multiplication is widely used in solution of computational problems. Despite its simplicity, the arithmetic complexity and data dependencies make it difficult to reduce its *run-time complexity.* The two basic approaches to decrease the run-time of matrix multiplication are: (1) *reducing the number of scalar multiplications*, while increasing the number of scalar additions/subtractions (and introducing irregularity of data access, as well as need for extra memory; see, discussion in [1]), and (2) *parallel implementation* of matrix multiplication (see, for example [2] and references found there). Of course, a combination of these two approaches is also possible (see discussion and references in [3–5]).

The (recursive) matrix multiplication "worked well" in theoretical analysis of arithmetical complexity, and when implemented on early computers. However, its implementation started to became a problem on computers with hierarchical memory (e.g. to reach optimal performance of a Strassen-type algorithm, recursion had to be stopped when the size of divided submatrices approximated the size of cache memory—differing between machines; see, [6, 7]), which contradicts the very idea of recursion. Furthermore, practical implementation of Strassen-type algorithms requires extra memory (e.g. Cray's implementation Strassen's algorithm required extra space of order $2.34 \cdot n^2$). The situation became even more complex when parallel Strassen-type algorithms have been implemented [5]. Interestingly, the research in recursive matrix multiplication seems to be diminishing, with the last paper known to the authors' published in 2006.

© Springer-Verlag Berlin Heidelberg 2012

In addition to the standard (linear-algebraic) matrix multiplication, a more *general* matrix multiplication appears as a kernel of algorithms solving the Algebraic Path Problem (APP). Here, the examples are: finding the all-pairs shortest paths, finding the most reliable paths, etc. In most of them, a generalized form of a $C \leftarrow C \oplus A \otimes B$ matrix operation plays a crucial role in finding the solution. This, generalized, matrix multiplication is based on the algebraic theory of semirings (for an overview of computational issues in the APP, and their relation to the theory of semirings, see [8], and references collected there). Note that standard linear algebra (with its matrix multiplication) is just one of the examples of algebraic (matrix) semirings.

While algebraic semirings can be seen as a simple "unification through generalization" of a large class of computational problems, they should be viewed in the context of ongoing changes in computer (processor) architectures. Specifically, the success of fused multiply-and-add (FMA) units, which take three scalar (in) operands and produce a single (out) result within a single clock cycle. Furthermore, GPU processors from Nvidia and AMD combine multiple FMA units (e.g. the Nvidia's Fermi chip allows 512 single-precision FMA operations completed in a single cycle; here, we omit issues concerning the data-feed bottleneck).

Our aim is to combine: (a) fast matrix multiplication, (b) mathematics of semirings, and (c) trends in computer hardware, to propose a generalized matrix multiplication, which can be used to develop efficient APP solvers.

2 Algebraic Semirings in Scientific Calculations

Since 1970's, a large number of problems has been combined under a single umbrella, named the *Algebraic Path Problem* (*APP*; see [9]). Furthermore, it was established that the matrix multiply-and-add (MMA) operations, in different algebraic semirings, are used as a centerpiece of various APP solvers.

A closed semiring $(S, \oplus, \otimes, *, \bar{0}, \bar{1})$ is an algebraic structure defined for a set S, with two binary operations: addition $\oplus : S \times S \rightarrow S$ and multiplication $\otimes : S \times S \rightarrow S$, a unary operation called *closure* $\circledast : S \rightarrow S$, and two constants $\bar{0}$ and $\bar{1}$ in S. Here, we are particularly interested in the set S consisting of matrices. Thus, following [9], we introduce a matrix semiring $(S^{n\times n}, \bigoplus, \bigotimes, \bigstar, \bar{O}, \bar{I})$ as a set of $n \times n$ matrices $S^{n\times n}$ over a closed scalar semiring $(S, \oplus, \otimes, *, \bar{0}, \bar{1})$ with two binary operations, matrix addition $\bigoplus : S^{n\times n} \times S^{n\times n} \rightarrow S^{n\times n}$ and matrix multiplication $\bigotimes : S^{n\times n} \times S^{n\times n} \rightarrow S^{n\times n}$, a unary operation called *closure of a matrix* $\bigstar : S^{n\times n} \rightarrow S^{n\times n}$, the zero $n \times n$ matrix $\bar{O}$ whose all elements equal to $\bar{0}$, and the $n \times n$ identity matrix $\bar{I}$ whose all main diagonal elements equal to $\bar{1}$ and $\bar{0}$ otherwise. Here, matrix addition and multiplication are defined as usually in linear algebra. Note that the case of rectangular matrices can be dealt with satisfactorily, and is omitted for clarity of presentation.

As stated, large number of matrix semirings appear in well-studied APP's. We summarize some of them in a table (similar to that presented in [10]). For simplicity of notation, in the Table 1, we represent them in the scalar form. Note that the *Minimum reliability path* problem has not been encountered by

Table 1. Semirings for various APP problems

S	$\oplus$	$\otimes$	$\circledast$	$\bar{0}$	$\bar{1}$	Application
$(0,1)$	$\vee$	$\wedge$	1	0	1	Transitive and reflexive closure of binary relations
$\mathbb{R}\cup+\infty$	$+$	$\times$	$1/(1-r)$	0	1	Matrix inversion
$\mathbb{R}_+\cup+\infty$	min	$+$	0	∞	0	All-pairs shortest paths problem
$\mathbb{R}_+\cup-\infty$	max	$+$	0	-∞	0	Maximum cost (critical path)
$[0,1]$	max	$\times$	1	0	1	Maximum reliability paths
$[0,1]$	min	$\times$	1	0	1	Minimum reliability paths
$\mathbb{R}_+\cup+\infty$	min	max	0	∞	0	Minimum spanning tree
$\mathbb{R}_+\cup-\infty$	max	min	0	-∞	0	Maximum capacity paths

the authors before. It was defined on the basis of systematically representing possible semirings—as a natural counterpart to the *Maximum reliability problem* (the only difference is the $\oplus$ operation: *min* instead of *max*). Since the maximum reliability path defines the *best* way to travel between two vertices of a graph; the *Minimum reliability problem* could be interpreted as: finding the *worst* pathway, one that should not be "stepped into").

While Table 1 summarizes the *scalar* semirings, and *scalar* operations, kernels of blocked algorithms for solving the APP, are based on (block) MMA operations [11]. Therefore, let us present the relation between the scalar multiply-and-add operation (ω), and the corresponding MMA kernel (α), for semirings in Table 1 (here, Nb is the size of a matrix block; see, also [12]):

- Matrix Inversion Problem:
  (α) $a(i,j) = a(i,j) + \sum_{k=0}^{Nb-1} a(i,k) \times a(k,j)$;
  (ω) $c = a \times b + c$;
- All-Pairs Shortest Paths Problem:
  (α) $a(i,j) = \min\big\{a(i,j), \min_{k=0}^{Nb-1}[a(i,k) + a(k,j)]\big\}$;
  (ω) $c = \min(c, a+b)$;
- All-Pairs Longest (Critical) Paths Problem:
  (α) $a(i,j) = \max\big\{a(i,j), \max_{k=0}^{Nb-1}[a(i,k) + a(k,j)]\big\}$;
  (ω) $c = \max(c, a+b)$;
- Maximum Capacity Paths Problem:
  (α) $a(i,j) = \max\big\{a(i,j), \max_{k=0}^{Nb-1} \min[a(i,k), a(k,j)]\big\}$;
  (ω) $c = \max[c, \min(a,b)]$;
- Maximum Reliability Paths Problem:
  (α) $a(i,j) = \max\big\{a(i,j), \max_{k=0}^{Nb-1}[a(i,k) \times a(k,j)]\big\}$;
  (ω) $c = \max(c, a \times b)$;
- Minimum Reliability Paths Problem:
  (α) $a(i,j) = \min\big\{a(i,j), \min_{k=0}^{Nb-1}[a(i,k) \times a(k,j)]\big\}$;
  (ω) $c = \min(c, a \times b)$;
- Minimum Spanning Tree Problem:
  (α) $a(i,j) = \min\big\{a(i,j), \min_{k=0}^{Nb-1}[\max\big(a(i,k), a(k,j)\big)]\big\}$;
  (ω) $c = \min[c, \max(a,b)]$.

Summarizing, the *generalized* MMA is one of the key operations for solving APP problems, and a large class of standard MMA-based numerical linear algebraic algorithms. Note that, this latter class includes most of block-oriented formulations of standard problems involving the level 3 BLAS operations [13].

3 Matrix Operations and Computer Hardware: Today and in the Near Future

In the late 1970's it was realized that many algorithms for matrix computations consist of similar building blocks (e.g. a *vector update*, or a *dot-product*). As a result, first, during the design of the Cray-1 supercomputer, vector operations of type $y \leftarrow y + \alpha \cdot x$ (where x and y are n-element vectors, while α is a scalar) have been efficiently implemented in the hardware. Specifically, vector processors have been designed with *chaining* of multiply-and-add operations [14]. Here, results of the multiply operations have been forwarded directly from the multiplication unit to the addition unit. Second, in 1979, the (level 1) BLAS standard was proposed [15], which defined, a set of vector operations. Here, it was assumed that computer vendors would provide their efficient hardware (and software) realizations. In this spirit, Cray Inc. built computers with efficient *vector updates*, and developed the *scilib* library; while the IBM build the ES/9000 vector computers with efficient *dot-product* operation, and developed the *ESSL* library. This library was later ported to the IBM RS/6000 workstations; the first commercial computer to implement the fused multiply-add (FMA) operation [16]). Following this path, most of today advanced processors from IBM, Intel, AMD, Nvidia, and others include scalar floating-point fused multiply-add (FMA) instruction.

The FMA operation combines two basic floating-point operations (flops) into one (three-read-one-write) operation with only one rounding error, throughput of two flops per cycle, and a few cycles latency – depending on the depth of the FMA pipeline. Besides the increased accuracy, the FMA minimizes operation latency, reduces hardware cost, and chip busing [16]. The standard floating-point add, or multiply, are performed by taking $a = 1.0$ (or $b = 1.0$) for addition, or $c = 0.0$ for multiplication. Therefore, the two floating-point constants, 0.0 and 1.0, are needed (and made available within the processor).

FMA-based kernels speed-up ($\sim 2\times$) solution of many scientific, engineering, and multimedia problems, which are based on the linear algebra (matrix) transforms [17]. However, other APP problems suffer from lack of hardware support for the needed scalar FMA operations (see, Table 1). The need for min or/and max operations in the generalized MMA operations introduces one or two conditional branches or comparison/selection instructions, which are highly undesirable for deeply pipelined processors. Recall that each of these operations is repeated Nb-times in the corresponding kernel (see, Section 2), while the kernel itself is called multiple times in the blocked APP algorithm. Here, note the recent results (see, [12]), concerning evaluation of the MMA operation in different semirings, on the Cell/B.E. processor. They showed that the "penalty" for lack of the *generalized* FMA unit may be up to 400%. This can be also viewed as: having an FMA unit, capable of supporting operations and special elements from Table 1 could speed-up solution of APP problems by up to 4 times.

Interestingly, we have just found that the AMD Cypress GPU processor supports the $(\min, \max)$-operation through a single call with 2 clock cycles per result. In this case, the Minimum Spanning Tree (MSP) problem (see, Table 1) could be solved more efficiently than previously realized. Furthermore, this could mean that the AMD hardware has $-\infty$ and ∞ constants already build-in. This, in turn, could constitute an important step towards hardware support of generalized FMA operations, needed to realize all kernels listed in Table 1.

Let us now look into the recent trends in *parallel hardware.* In the 1990's three main designs for parallel computers were: (1) array processors, (2) shared memory parallel computers, and (3) distributed memory parallel computers. After a period of dormancy, currently we observe a resurgence of array-processor-like hardware, placed within a single processor. In particular, the Nvidia promises processors consisting of thousands of computational (FMA) units. This perspective has already started to influence the way we write codes. For instance, one of the key issues is likely to become (again) the *simplicity and uniformity of data manipulation*, while accepting the price of performing seemingly unnecessary operations. As an example, for sparse matrix operations, the guideline can be the statement made recently by John Gustafson, who said: "Go Ahead, Multiply by Zero!" [18]. His assumption, like ours, is that in the hardware of the future, cost of an FMA will be so low, in comparison with data movement (and indexing), that computational sparse linear algebra will have to be re-evaluated.

4 Proposed Generalized Multiply-Add Operation

Let us now summarize the main points made thus far. First, the future of efficient parallel MMA is not likely to involve recursion, focused on reduction the total number of scalar multiplications, while not paying attention to the cost of data movement (and extra memory). Second, without realizing this, scientists solving large number of computational problems, have been working with algebraic semirings. Algebra of semirings involves not only standard linear algebra, but also a large class of APP's. Solutions to these problems involve generalized MMA (which, in turn, calls for hardware-supported generalized FMA operations). Third, benefits of development of *generalized FMA units*, capable of dealing with operations listed in Table 1 have been illustrated. Finally, we have recalled that current trends of development of computer hardware point to existence of processors with thousands of FMA units (possibly generalized), similar to SIMD computers on the chip. Based on these considerations, we can define the needed generic matrix multiply-and-add (MMA) operation

$$\mathtt{C} \leftarrow \mathtt{MMA}[\otimes, \oplus](\mathtt{A}, \mathtt{B}, \mathtt{C}) : \mathtt{C} \leftarrow \mathtt{A} \otimes \mathtt{B} \oplus \mathtt{C},$$

where the $[\otimes, \oplus]$ operations originate from different matrix semirings. Note that, like in the scalar FMA operations, generalized matrix *addition* or *multiplication*, can be implemented by making an $n \times n$ matrix $\mathtt{A}$ (or $\mathtt{B}$) $= \bar{O}$ for addition, or a matrix $\mathtt{C} = \bar{I}$ for multiplication (where the appropriate zero and identity matrices have been defined in Section 2).

Observe that the proposed generalization allows a new approach to the development of efficient codes solving a number of problems. On the one hand, it places in the right context (and subsumes) the level 3 BLAS matrix multiplication (more in Section 5). On the other, the same concepts and representations of operations can be used in solvers for the APP problems.

Let us stress that the idea is not only to generalize matrix multiplication via application of algebraic semirings, but also to "step-up" use of the MMA as the main operation for matrix algorithms. In this way, we should be able (among others) to support processors with thousands of FMA cores. To illustrate the point, let us show how generalized matrix multiplication can be used to perform selected auxiliary matrix operations.

4.1 Data Manipulation by Matrix Multiplication

Observe that the MMA operation, which represents a linear transformation of a vector space, can be used not only for computing but also for data manipulations, such as: reordering of matrix rows/columns, matrix rotation, transposition, etc. This technique is well established and widely used in the algebraic theory (see, for example, [19]). Obviously, data manipulation is an integral part of a large class of matrix algorithms, regardless of parallelism. Due to the space limitation, let us show only how the MMA operation can be used for classical matrix data manipulations, like the row/column interchange, and how it can be extended for more complex data transforms, like th global reduction and replication (broadcast).

Row/Column Interchange. If `zeros(n,n)` is an $n \times n$ zero matrix and `P(n,n)` is an $n \times n$ permutation matrix obtained by permuting the i-th and j-th rows of the identity matrix $I_{n\times n} = \mathtt{eye(n,n)}$ with $i < j$, then multiplication

$$\mathtt{D(n,n)} = \mathtt{MMA}[\times,+]\big(\mathtt{P(n,n)},\mathtt{A(n,n)},\mathtt{zeros(n,n)}\big)$$

gives a matrix `D(n,n)` with the i-th and j-th rows of `A(n,n)` interchanged, and

$$\mathtt{D(n,n)} = \mathtt{MMA}[\times,+]\big(\mathtt{A(n,n)},\mathtt{P(n,n)},\mathtt{zeros(n,n)}\big)$$

gives `D(n,n)` with the i-th and j-th columns of `A(n,n)` interchanged.

Global Reduction and Broadcast. A combination of the global reduction and broadcast, also known in the MPI library as the `MPI_ALLREDUCE` [20], is a very important operation in parallel processing. It is so important that, for example, in the IBM BlueGene/L, a special *collective* network is used, in addition to the other available communication networks, including a 3D toroidal network [21]. However, this operation can be also represented as three matrix multiplications in different semirings,

$$\mathtt{B(n,n)} = \mathtt{ones(n,n)} \otimes \mathtt{A(n,n)} \otimes \mathtt{ones(n,n)},$$

where $\mathtt{ones(n,n)}$ is the $n \times n$ matrix of ones. For example, the reduction (summation) of all elements of a matrix $\mathtt{A}$ to the single scalar element and its replication (broadcast) to the matrix $\mathtt{B}$ can be completed by two consecutive MMA operations in the $(\times, +)$-semiring:

$$\mathtt{C(n,n)} = \mathtt{MMA}[\times,+]\big(\mathtt{ones(n,n)}, \mathtt{A(n,n)}, \mathtt{zeros(n,n)}\big),$$

for summation of elements along columns of a matrix $\mathtt{A(n,n)}$, and then,

$$\mathtt{B(n,n)} = \mathtt{MMA}[\times,+]\big(\mathtt{C(n,n)}, \mathtt{ones(n,n)}, \mathtt{zeros(n,n)}\big),$$

for summation of elements along the rows of an intermediate matrix $\mathtt{C}$. For a sample 4×4 matrix $\mathtt{A}$, it will be completed as follows

$$\begin{pmatrix} 1\,1\,1\,1 \\ 1\,1\,1\,1 \\ 1\,1\,1\,1 \\ 1\,1\,1\,1 \end{pmatrix} \times \begin{pmatrix} 1\,2\,3\,4 \\ 5\,6\,7\,8 \\ 4\,3\,2\,1 \\ 8\,7\,6\,5 \end{pmatrix} \times \begin{pmatrix} 1\,1\,1\,1 \\ 1\,1\,1\,1 \\ 1\,1\,1\,1 \\ 1\,1\,1\,1 \end{pmatrix} = \begin{pmatrix} 72\;72\;72\;72 \\ 72\;72\;72\;72 \\ 72\;72\;72\;72 \\ 72\;72\;72\;72 \end{pmatrix}.$$

If the $(\times, \max)$-semiring is used in these operations, i.e. the scalar multiply-and-add operation would be $c = \max(a \times b, c)$, then the maximal element selection and its broadcast can be implemented as

$$\mathtt{C(n,n)} = \mathtt{MMA}[\times,\max]\big(\mathtt{ones(n,n)}, \mathtt{A(n,n)}, -\mathtt{inf(n,n)}\big),$$

$$\mathtt{B(n,n)} = \mathtt{MMA}[\times,\max]\big(\mathtt{C(n,n)}, \mathtt{ones(n,n)}, -\mathtt{inf(n,n)}\big),$$

where $-\mathtt{inf(n,n)}$ is an $n \times n$ matrix of negative infinity. Thus, we have

$$\begin{pmatrix} 1\,1\,1\,1 \\ 1\,1\,1\,1 \\ 1\,1\,1\,1 \\ 1\,1\,1\,1 \end{pmatrix} \times \begin{pmatrix} 1\,2\,3\,4 \\ 5\,6\,7\,8 \\ 4\,3\,2\,1 \\ 8\,7\,6\,5 \end{pmatrix} \times \begin{pmatrix} 1\,1\,1\,1 \\ 1\,1\,1\,1 \\ 1\,1\,1\,1 \\ 1\,1\,1\,1 \end{pmatrix} = \begin{pmatrix} 8\,8\,8\,8 \\ 8\,8\,8\,8 \\ 8\,8\,8\,8 \\ 8\,8\,8\,8 \end{pmatrix}.$$

It is clear that the implementation of this operation in the $(\times, \min)$-semiring with a matrix $\mathtt{inf(n,n)}$ will select the minimal element in the matrix $\mathtt{A(n,n)}$ and its replication, or broadcast.

Interestingly, all these (and other possible) operations for a matrix data manipulation can be realized on an $n \times n$ torus array processor by using a single, or multiple, MMA operation(s), *without any* additional interconnection networks (see, [22]). For implementation of each $n \times n$ MMA operation on the torus array processor, n time-steps are needed and, therefore, only $2n$ steps are required to realize the global reduction and broadcast.

Note that basic elementary matrices, like $\mathtt{eye(n,n)}$, $\mathtt{zeros(n,n)}$, $\mathtt{ones(n,n)}$, and $\pm\mathtt{inf(n,n)}$, can be hardwired within an array processor, where each of their elements is stored in the corresponding processing element. The other required transform matrices can be formed within an array processor by using these elementary matrices, and the corresponding MMA operations. In fact, as stated above, todays' advanced microprocessors already contain special registers for storing the fundamental constants like floating-point 0.0, 1.0, or the $\pm\infty$. An array of such microprocessors will, therefore, automatically include some of the needed elementary matrices.

5 Relating BLAS to the Generalized MMA

In the final part of this paper, let us briefly look into the interrelation between the proposed generalized MMA and the BLAS standard [15, 23, 13]. Due to the lack of space we focus our attention on the level 3 BLAS [13] (the remaining two levels require addressing some subtle points, and will be discussed separately). For similar reasons, we also restrict our attention to square non-symmetric matrices. However, these simplifications do not diminish generality of our conclusions.

In our work we are concerned with the generalized MMA in the form: $C \leftarrow C \oplus A \otimes B$. This matches directly the definition of the level 3 BLAS _GEMM routine, which performs operation: $C \leftarrow \beta C + \alpha op(A) \times op(B)$, while its arguments are:

$$\text{\_GEMM}(\texttt{transa}, \texttt{transb}, \texttt{m}, \texttt{n}, \texttt{k}, \alpha, \texttt{A}, \texttt{lda}, \texttt{B}, \texttt{ldb}, \beta, \texttt{C}, \texttt{ldc}).$$

When using _GEMM in computations, the "_" is replaced by one of the letters S, D, C, Z defining the type of matrices (real, double real, complex, and double complex, respectively); while the transa and the transb specify if matrices A and B are in the standard or in the transposed form.

Let us now relate the semiring-based MMA and the _GEMM. To use the semiring constructs in computations, we have to distinguish two aspects. First, the information that is needed to specify the unique semiring and, second, specification of operands of operations within that semiring. As discussed in Section 2, to define a semiring we need to specify: (1) elements (S), (2) operations ($\oplus, \otimes, \circledast$), and (3) special elements ($\bar{0}$ and $\bar{1}$), Obviously, the semiring defined by the BLAS operations is the linear algebra semiring. Here, the only aspect of the semiring that is *explicitly* defined, is the set of elements of the matrices. This definition comes through the "naming convention" and is realized by the "_" part of the subroutine definition; selecting the type of the numbers (real, or complex) and their computer representation (single, or double precision). The remaining parts of the definition are implicit. It is assumed that the objects of the semiring are matrices, while the elements $\bar{0}$ and $\bar{1}$ are matrices consisting of all zeros, and the standard identity matrix, respectively. Here, operations are the basic matrix multiplication, addition and closure. At the same time, the elements m, n, k, α, A, lda, B, ldb, β, C, ldc define specific operands for the MMA.

An interesting issue concerns the transa and transb parameters. They specify if matrices A and B (respectively), used in the _GEMM operation are in their standard or transposed forms. From the point of view of the theory of semirings, it does not matter if a matrix is transposed or not; what matters is if it belongs to the set S. The difference occurs when the actual MMA is to be performed. Therefore, the standard BLAS operation set allowed to combine matrix multiplication, scaling, and transpose operations into a single, relatively simple, code. However, as illustrated in Section 4.1, in the case of the approach advocated in this paper, where "everything is a matrix multiplication," matrix scaling and transpose also become matrix multiplications.

It should be noted that all, but three, remaining level 3 BLAS routines also perform operations of the type $C = \beta C + \alpha op(A) op(B)$, for a variety of specific operands. The only exceptions are the _HER2K, _SYR2K pair, which perform "double updates" of a matrix. Obviously they can be replaced by two operations of the type $C = \beta C + \alpha op(A)op(B)$ performed in a row (two MMA's).

The only operation that is not easily conceptualized, within the scope of the proposal outlined thus far, is the _TRSM which performs a triangular solve. Here the matrix inversion operation (defined in Table 1), could be utilized; but the requires further careful considerations.

6 Concluding Remarks

In this paper we have reflected on the effect that the recent developments in computer hardware and computational sciences can have on the way that dense matrix multiplication is approached. First, we have indicated that the arithmetical operation count (theoretical floating point complexity) looses importance; becoming overshadowed by the complexity of data manipulations performed by processors with hundreds of FMA processing units. Second, we have recalled that the "standard" matrix multiplication is just one of possible operations within an appropriately defined algebraic semiring. This allowed us to illustrate how the semiring-based approach covers large class of APP problems. Finally, on the basis of these considerations, we have proposed a generalized matrix multiply-and-add operation, which allows to further induce efficient matrix multiplication as the key operation driving solution methods not only in linear algebra, but also across a variety of APP problems. Finally, we have outlined the relation of the proposed matrix generalization to the level 3 BLAS standard.

In the near future we plan to (1) propose an object oriented model for the generalized MMA (see, [24] for description of initial work in this direction), (2) proceed with a prototype implementation, and (3) conduce experiments with the, newly-defined, fused multiply-add operations involving *min* and *max* operations.

Acknowledgment. Work of Marcin Paprzycki was concluded while visiting the University of Aizu as a Research Scientist.

References

1. Robinson, S.: Towards an optimal algorithm for matrix multiplication. SIAM News 38 (2005)
2. Li, J., Skjellum, A., Falgout, R.D.: A poly-algorithm for parallel dense matrix multiplication on two-dimensional process grid topologies. Concurrency - Practice and Experience 9, 345–389 (1997)
3. Hunold, S., Rauber, T., Rünger, G.: Combining building blocks for parallel multi-level matrix multiplication. Parallel Comput. 34, 411–426 (2008)
4. Grayson, B., Van De Geijn, R.: A high performance parallel Strassen implementation. Parallel Processing Letters 6, 3–12 (1996)
5. Song, F., Moore, S., Dongarra, J.: Experiments with Strassen's Algorithm: from Sequential to Parallel. In: International Conference on Parallel and Distributed Computing and Systems (PDCS 2006). ACTA Press (November 2006)

6. Bailey, D.H., Lee, K., Simon, H.D.: Using Strassen's algorithm to accelerate the solution of linear systems. J. Supercomputer 4, 357–371 (1991)
7. Paprzycki, M., Cyphers, C.: Multiplying matrices on the Cray – practical considerations. CHPC Newsletter 6, 77–82 (1991)
8. Sedukhin, S.G., Miyazaki, T., Kuroda, K.: Orbital systolic algorithms and array processors for solution of the algebraic path problem. IEICE Trans. on Information and Systems E93.D, 534–541 (2010)
9. Lehmann, D.J.: Algebraic structures for transitive closure. Theoretical Computer Science 4, 59–76 (1977)
10. Abdali, S.K., Saunders, B.D.: Transitive closure and related semiring properties via eliminants. Theoretical Computer Science 40, 257–274 (1985)
11. Matsumoto, K., Sedukhin, S.G.: A solution of the all-pairs shortest paths problem on the Cell broadband engine processor. IEICE Trans. on Information and Systems 92-D, 1225–1231 (2009)
12. Sedukhin, S.G., Miyazaki, T.: Rapid*Closure: Algebraic extensions of a scalar multiply-add operation. In: Philips, T. (ed.) CATA, ISCA, pp. 19–24 (2010)
13. Dongarra, J.J., Croz, J.D., Duff, I., Hammarling, S.: A set of level 3 basic linear algebra subprograms. ACM Trans. Math. Software 16, 1–17 (1990)
14. Russell, R.M.: The CRAY-1 computer system. Commun. ACM 21, 63–72 (1978)
15. Lawson, C.L., Hanson, R.J., Kincaid, R.J., Krogh, F.T.: Basic linear algebra subprograms for FORTRAN usage. ACM Trans. Math. Software 5, 308–323 (1979)
16. Montoye, R.K., Hokenek, E., Runyon, S.L.: Design of the IBM RISC System/6000 floating-point execution unit. IBM J. Res. Dev. 34, 59–70 (1990)
17. Gustavson, F.G., Moreira, J.E., Enenkel, R.F.: The fused multiply-add instruction leads to algorithms for extended-precision floating point: applications to Java and high-performance computing. In: CASCON 1999: Proceedings of the 1999 Conference of the Centre for Advanced Studies on Collab. Research, p. 4. IBM Press (1999)
18. Gustafson, J.L.: Algorithm leadership. HPCwire, April 06 (2007)
19. Birkhoff, G., McLane, S.: A Survey of Modern Algebra. AKP Classics. A K Peters, Massachusetts (1997)
20. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI: The Complete Reference. The MIT Press, Cambridge (1996)
21. Gara, A., Blumrich, M.A., Chen, D., Chiu, G.L.T., Coteus, P., Giampapa, M., Haring, R.A., Heidelberger, P., Hoenicke, D., Kopcsay, G.V., Liebsch, T.A., Ohmacht, M., Steinmacher-Burow, B.D., Takken, T., Vranas, P.: Overview of the Blue Gene/L system architecture. IBM J. Res. and Dev. 49, 195–212 (2005)
22. Sedukhin, S.G., Zekri, A.S., Myiazaki, T.: Orbital algorithms and unified array processor for computing 2D separable transforms. In: International Conference on Parallel Processing Workshops, pp. 127–134 (2010)
23. Dongarra, J.J., Croz, J.D., Hammarling, S., Hanson, R.J.: An extended set of FORTRAN basic linear algebra subprograms. ACM Trans. Math. Software 14, 1–17 (1988)
24. Ganzha, M., Sedukhin, S., Paprzycki, M.: Object oriented model of generalized matrix multipication. In: Proceedings of the Federated Conference on Computer Science and Information Systems, pp. 439–442. IEEE Press, Los Alamitos (2011)