Analysis of data flow for SIMD systems

Reinhard Klette

0. Introduction

A general approach to characterizing the inherent complexity of computational problems is given by the quantitative analysis of the extent of the data flow that has to be performed during the solution of these problems. On the other hand, any parallel processing system possesses a restricted ability for fast data transfer determined essentially by the interconnection pattern of the processing elements. In the present paper, these general observations, as previously mentioned by Gentleman (1978), Siegel (1979), Abelson (1980), or Klette (1980), will be transformed into precise definitions of local, global and total data transfer within SIMD systems, and the corresponding definitions of local, global and total data dependencies for computational problems as well. The basic relation between these corresponding notions — the computational time must at least be sufficient for realizing the necessary extent of data transfer — will be represented in a so-called data transfer lemma that outlines the starting point of our formalized method of obtaining lower time bounds by data flow analysis. This approach will be illustrated by application to a variety of different parallel processing architectures where the unifying feature will be that we shall use SIMD models that employ an interconnection network and use no shared memory. Our parallel processing systems will be abstract models of computation where the level of abstraction may be compared with that of a random access machine (RAM); cp. Aho et al. [2] for this model of serial computation. For computational problems such as those mentioned in the present paper the author was inspired by the digital image processing area, where reference is made to Rosenfeld et al. [9] and Klette [5]. But, of course, this does not represent a serious restriction; e.g., matrix multiplication or pattern matching are computational problems of general importance.

The general SIMD model as used in this paper is characterized by a finite or infinite set of processing elements (PEs), an interconnection network, and a central processing unit (CPU). For a rough scheme of an SIMD system which the reader may have in mind throughout this paper, see Fig. 1.

CPU. The CPU has a (central) random access memory which consists of a finite or infinite sequence of registers $r_0, r_1, r_2, \ldots$ with a distinguished accumu-

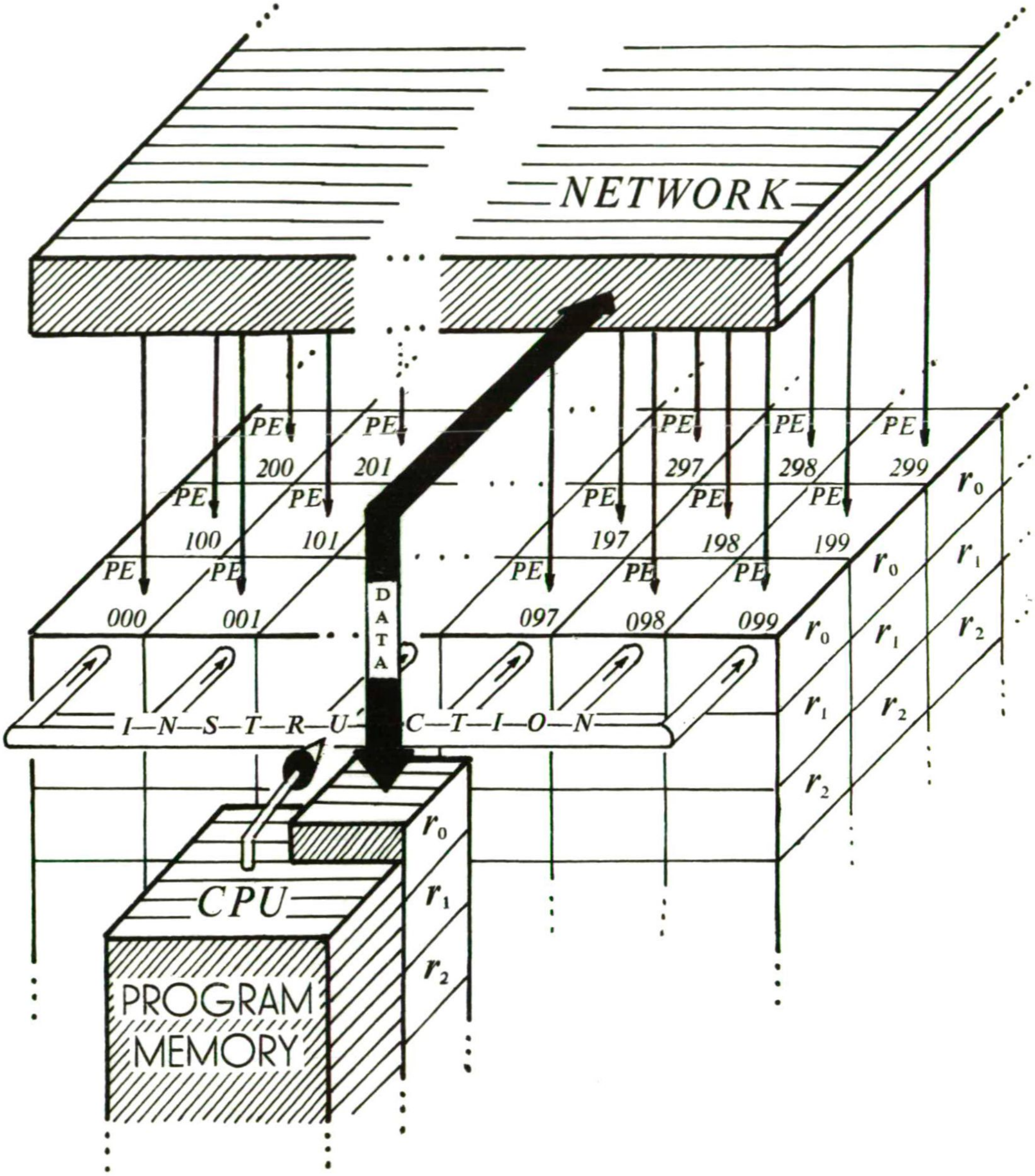

Figure 1.
Scheme of an SIMD system

lator r_0. Let D_{CPU} be the depth of this random access memory, i.e., the number of CPU registers, for $1 \leqq D_{\mathrm{CPU}} \leqq \infty$. Furthermore, let W_{CPU} be the word length of these registers (number of bit positions), which is assumed to be constant for all CPU registers, for $1 \leqq W_{\mathrm{CPU}} \leqq \infty$. The CPU spreads a single instruction stream to the synchronized working PEs. The programs of the system are stored in a, potentially size-unlimited, special program memory of the CPU. Part of any instruction addressed to the PEs is an enable/disable mask to select a subset of the PEs that are to perform the given instruction; the remaining PEs will be idle. The CPU may read the accumulator contents of any one PE of a specified subset of all PEs, and is able to transfer its accumulator contents to some of the PE accumulators. Any data transfer between CPU and PEs is restricted to serial mode.

PEs. Each PE has some (local) random access memory which consists of a finite or infinite sequence of registers $r_0, r_1, r_2, \ldots$ with a distinguished register r_0 called the accumulator. Let D_{PE} be the depth of these random access memories, i.e., this depth is assumed to be constant for all PEs of a given system, for $1 \leqq D_{\mathrm{PE}} \leqq \infty$. Furthermore, let W_{PE} be the unique word length of the PE registers, for $1 \leqq W_{\mathrm{PE}} \leqq \infty$. Each PE is capable of performing some basic operations which take place in its accumulator. Direct data access is restricted to its own registers, to the accumulators of the directly connected PEs in the sense of the given interconnection network, and, possibly, to the accumulator of the CPU. The PEs are indexed by integers or tuples of integers. Each PE knows its index. Let N_{PE}, $0 \leqq N_{\mathrm{PE}} \leqq \infty$, be the number of PEs of a given system, and $\mathbf{ind} = \{j_1, j_2, \ldots, j_{N_{\mathrm{PE}}}\}$ be the set of all PE indices of a given SIMD system.

Interconnection network. Each PE is located in a node of a given undirected graph representing the two-way interconnection scheme. Any PE may uniquely identify the different edges connected to its node by using a given coding scheme. Let N_{IN} be the branching degree of the network, i.e., the maximum degree of the nodes of the given graph, for $0 \leqq N_{\mathrm{IN}} < \infty$.

For the selection of a specialized SIMD model the following system features may be concretely specified:

- off-line or on-line communication with the outside world,
- special values for $N_{\mathrm{PE}}, N_{\mathrm{IN}}, D_{\mathrm{CPU}}, D_{\mathrm{PE}}, W_{\mathrm{CPU}}$, or W_{PE},
- the set **ind**,
- the interconnection network structure including the edge coding scheme,
- the CPU instruction set including the available set of enable/disable masks as well as the method of the data exchange between CPU and PEs, and
- the restrictions on the system in communication with the outside world, i.e., input and output management.

Note that as regards the technical realization of an SIMD computing facility, in principle, one implementation may offer different ways to run such a system, i.e., the working principles of several SIMD models as considered in the present paper may be unified within one implementation. Essentially, this is the problem of constructing a flexible interconnection network with reconfigurability, and/or of running a system using different modes.

The outline of this paper is as follows. In the first section we shall present some standardized system description features for specifications of SIMD models. In Section 2 we shall describe how the data flow of an SIMD system may be measured by functions in a quantitative way. Then, in Section 3 the corresponding notions of data dependencies will be explained for computational problems. In Section 4 the data transfer lemma will be given as well as some applications of this lemma to different models of computation for lower time bound determination. Our concluding remarks are given at the end of the paper.

The standard SIMD models as described in Section 1 constitute the framework of a parallel simulation system (PARSIS) presently under implementation; cp. Legendi [7] for a similar project for simulation of cellular processors.

1. OFF-NETs and ON-NETs

In our experience in parallel program design the exclusion of given technical restrictions, e.g., on $N_{\mathrm{PE}}, N_{\mathrm{IN}}$, etc., in the first steps of problem solutions, enables us to find important methods of parallelization of solution processes as well as general features for system description. Of course, for concrete implementation quite a lot of time must be spent in taking given restrictions for $N_{\mathrm{PE}}, N_{\mathrm{IN}}$, etc. into consideration. The present paper is concerned with the first phase, the theoretical preparation for the second phase, which is the concrete implementation. In this sense, we shall deal with abstract SIMD models throughout this paper. More detailed discussion will be the subject of forthcoming papers, depending on the progress of the PARSIS project.

The common one-accumulator computer, e.g., the random access machine (RAM) in the sense of Aho et al. [2], may be considered as the simplest example of an abstract SIMD system — $N_{\mathrm{PE}}=0$ and $D_{\mathrm{CPU}}=W_{\mathrm{CPU}}=\infty$. We shall use the RAM as the underlying model for serial data processing where, in distinction to [2], infinite precision, real number arithmetic is assumed, which is convenient for our theoretical considerations of computational problems such as the Fourier transform, or for operations on finite sets of points in the real plane, by avoiding discussions of round-off errors. In this sense, our standardized system description features start with the declaration of abstract registers.

Abstract registers. For an SIMD system with abstract registers we assume that any register may store one real number at a time, without any special encoding tricks. For our theoretical considerations in this paper, it is not important to specify how the reals are stored in these abstract registers by special bit representations.

Standard register enumeration. We assume a unique enumeration of all registers as follows. For registers r_m of the PE with index j or (j, k), called PE(j) or PE(j, k) in the sequel, we use the integer tuples (j, m) or (j, k, m), respectively, and for register r_m of the CPU just the integer m.

Uniform network structure. Either $N_{\mathrm{IN}}=0$, or $N_{\mathrm{IN}}=p\geqq 1$ and the network structure is characterized by p different functions $f_0, f_1, \dots, f_{p-1}$ on the set **ind** of all PE indices in the following way. For $j, k\in\mathbf{ind}$, PE(j) and PE(k) are *directly connected* iff there exists an i, $0\leqq i\leqq p-1$, such that $f_i(j)=k$. Because of our assumption that all connections are two-way it follows that

$$(\bigwedge j, k\in\mathbf{ind})[(\bigvee i\in\{0, 1, \dots, p-1\})f_i(j) = k \equiv (\bigvee h\in\{0, 1, \dots, p-1\})\ f_h(k) = j].$$

In [10] the functions $f_0, f_1, \dots, f_{p-1}$ were called *interconnection functions*. With the exception of a fixed set of PEs at the network border, we also claim that all

PEs are directly connected to exactly p different PEs. When $f_i(j)=k$, PE(k) is called the *ith neighbor* of PE(j). In this way, the edge coding scheme for uniform networks is defined. For each PE, the neighborhood consists of all (i.e., at most p) neighbor PEs. Examples of infinite networks as well as finite networks matching our uniformity demand are given in Table 1. In the sequel we shall use these networks as defined here.

Some remarks are necessary regarding Table 1. The left-right 2^i (LR2I) network and the left-right-up-down 2^i network (LRUD2I) network were used for vector machines in Pratt et al. [8] and Klette et al. [6], respectively, without the restriction by an integer m as stated in Table 1. Note that we have restricted ourselves to interconnection networks with finite branching degree. The special form of the set **ind** in the Quadtree network is determined by our standard PE address masking scheme as defined later on. The finite uniform networks mentioned in Table 1 were studied by Siegel [10] — the perfect shuffle (PS), the ILLIAC, the Cube, the plus-minus 2^i (PM2I), and the wrap-around plus-minus 2^i (WPM2I) network, with the modification that the PS network is an undirected graph to match our uniform network convention, i.e., for the PS network the inverse shuffle function was added in comparison to [10]. For $j\in\mathbf{ind}=\{0, 1, ..., 2^m-1\}$ let $a_{m-1} \dots a_1 a_0$ denote the binary representation of j and $\bar{a}_i$ denote the complement of a_i. Then

$$\mathrm{exch}\,(a_{m-1}\dots a_1 a_0) = a_{m-1}\dots a_1\bar{a}_0,$$

$$\mathrm{shuf}\,(a_{m-1}\dots a_1 a_0) = a_{m-2}\dots a_1 a_0 a_{m-1},$$

$$\mathrm{shuf}^{-1}\,(a_{m-1}\dots a_1 a_0) = a_0 a_{m-1}\dots a_2 a_1,$$

$$\mathrm{cube}_i\,(a_{m-1}\dots a_{i+1}a_i a_{i-1}\dots a_0) = a_{m-1}\dots a_{i+1}\bar{a}_i a_{i-1}\dots a_0,$$

$$\mathrm{WPM}_{+i}(a_{m-1}\dots a_i\dots a_0) = b_{m-1}\dots b_i\dots b_0,$$

where $b_{i-1} \dots b_0 b_{m-1} \dots b_{i+1} b_i=(a_{i-1} \dots a_0 a_{m-1} \dots a_{i+1} a_i)+1 \bmod 2^m$,

$$\mathrm{WPM}_{-i}\,(a_{m-1}\dots a_i\dots a_0) = b_{m-1}\dots b_i\dots b_0,$$

where $b_{i-1} \dots b_0 b_{m-1} \dots b_{i+1} b_i=(a_{i-1} \dots a_0 a_{m-1} \dots a_{i+1} a_i)-1 \bmod 2^m$, for $0\leqq i<m$ and $m\geqq 1$.

Standard PE *masking scheme.* As standard masks we shall use the simple bit patterns for PE indices as used, for example, in [10]. In the case of integer indices, a standard PE address mask is given by an arbitrary, non-empty word on the alphabet $\{0, 1, x\}$ enclosed by brackets, where x represents the "dont't care" situation. The only PEs that will be active are those whose address (i.e., index) matches the mask from right to left, where the indices are given in binary representation; 0 matches 0, 1 matches 1, and either 0 or 1 matches x. For example, by mask [x] all PE's are activated. For the representation of concrete standard masks within programs, etc. we take liberties such as [all PE's] instead of [x], or [odd PE's] instead of [1x] if the rightmost bit position is assumed to be the sign position. In the case of integer tuple indices, the standard PE address masks are arbitrary tuples of non-empty words on $\{0, 1, x\}$ enclosed by brackets. Note that for infinite networks as given in Table 1 any given PE address mask activates an infinite manifold of PE's. For example, the mask [0xx] applied to the bintree network will

Table 1. Uniform networks

Network	ind	N_{IN}	Case	Edge coding scheme							
				0	1	2	3	4	5	6	7
LINEAR	integers	2	all	$j-1$	$j+1$	—	—	—	—	—	—
LR2I^m	integers	$2m$	all	$f_{2i}(j)=j+2^i$ and $f_{2i+1}(j)=j-2^i$ for $0\leqq i<m$ and $m\geqq 2$							
BINTREE	positive integers	3	$j\geqq 2$	$\lfloor j/2 \rfloor$	—	—	—	—	—	—	—
			all	—	$2j$	$2j+1$	—	—	—	—	—
TRIANGLE	positive integers	5	$j\geqq 2$	$\lfloor j/2 \rfloor$	—	—	—	—	—	—	—
			all	—	$2j$	$2j+1$	—	—	—	—	—
			$j\neq 2^i$	—	—	—	$j-1$	—	—	—	—
			$j\neq 2^i-1$	—	—	—	—	$j+1$	—	—	—
QUADTREE	$\bigcup_{i=0}^{\infty} \cdot$ $\cdot \{4^i, \ldots, 2\cdot 4^i-1\}$	5	$j\geqq 4$	$\lfloor j/4 \rfloor$	—	—	—	—	—	—	—
			all	—	$4j$	$4j+1$	$4j+2$	$4j+3$	—	—	—
HEXAGONAL	tuples of integers	3	all	$(j,k-1)$	$(j,k+1)$	—	—	—	—	—	—
			$j+k$ even	—	—	$(j-1,k)$	—	—	—	—	—
			$j+k$ odd	—	—	$(j+1,k)$	—	—	—	—	—
SQUARE	tuples of integers	4	all	$(j,k-1)$	$(j,k+1)$	$(j-1,k)$	$(j+1,k)$	—	—	—	—
TRIAGONAL	tuples of integers	6	all	$(j,k-1)$	$(j,k+1)$	$(j-1,k)$	$(j+1,k)$	$(j-1,k-1)$	$(j+1,k+1)$	—	—
DIAGONAL	tuples of integers	8	all	$(j,k-1)$	$(j,k+1)$	$(j-1,k)$	$(j+1,k)$	$(j-1,k-1)$	$(j+1,k+1)$	$(j-1,k+1)$	$(j+1,k-1)$
LRUD2I^m	tuples of integers	$4m$	all	$f_{4i}(j,k)=(j+2^i,k)$, $f_{4i+1}(j,k)=(j-2^i,k)$, $f_{4i+2}(j,k)=(j,k+2^i)$, $f_{4i+3}(j,k)=(j,k-2^i)$, for $0\leqq i<m$ and $m\geqq 2$							
PS^m	$\{0,1,\ldots,2^m-1\}$	3	all	exch	shuf	shuf^{-1}	—	—	—	—	—
ILLIAC^m	$\{0,1,\ldots,2^m-1\}$	4	all	$+1 \bmod 2^m$	$-1 \bmod 2^m$	$+\frac{m}{2} \bmod 2^m$	$-\frac{m}{2} \bmod 2^m$	—	—	—	—
CUBE^m	$\{0,1,\ldots,2^{m-1}\}$	m	all	$f_i(j)=\text{cube}_i(j)$, for $0\leqq i<m$							
PM2I^m	$\{0,1,\ldots,2^{m-1}\}$	$2m$	all	$f_{2i}(j)=j+2^i \bmod 2^m$, $f_{2i+1}(j)=j-2^i \bmod 2^m$, for $0\leqq i<m$							
WPM2I^m	$\{0,1,\ldots,2^{m-1}\}$	$2m$	all	$f_{2i}(j)=\text{WPM}_{+i}(j)$, $f_{2i+1}(j)=\text{WPM}_{-i}(j)$, for $0\leqq i<m$							

activate the processing elements PE(2) and PE(3) on layer 1 of the bintree, disables layer 2, enables the first four PE's of layer 3, and so on, where the common binary representation of non-negative integers is assumed for the PE indices of the bintree network.

Abstract CPU *instruction set.* For any one of our theoretical SIMD systems, we shall assume that its CPU instruction set may be obtained by special interpretation and selection of the instructions of an abstract CPU instruction reservoir defined as follows. There are two different types of instructions, parallel instructions for activating some of the PEs, and serial instructions where the CPU itself is addressed for certain activity. Any *parallel instruction* consists of a PE address mask, an operation code (READ, WRITE, LOAD, STORE, OP, or $\mathrm{OP}_{l+1}, l \geqq 1$), and an operation address α where we shall use the standard register enumeration for explaining the meaning of these operation addresses. For the *serial instructions*, we assume branching instructions JUMP b, JGTZ b, JZERO b, JLTZ b (where b symbolizes an instruction number in a CPU program and the contents of the CPU accumulator are tested), the HALT instruction, and instructions consisting of an operation code (READ, WRITE, LOAD, STORE, OP_1, or OP_2). See Table 2 for the complete abstract CPU instruction set without jump and stop instructions.

Table 2. Abstract CPU instruction set without test and stop instructions

Instruction	Possible operation address α			
[mask] READ α	m;	$^*m$		
[mask] WRITE α	m;	$^*m$		
[mask] LOAD α	m;	$^*m$;	$: i$	
[mask] STORE α	m;	$^*m$;	$: i_1, i_2, \ldots, i_l$	
[mask] OP_1 α	m;	$^*m$;	$: i$	
[mask] OP_2 α	m;	$^*m$;	$: i$	
[mask] OP_{l+1}	$: i_1, i_2, \ldots, i_l$			
READ α	m;	$^*m$		
WRITE α	$=x$;	m;	$^*m$	
LOAD α	$=x$;	m;	$^*m$;	(j)
STORE α	m;	$^*m$;	(j)	
OP_1 α	$=x$;	m;	$^*m$;	(j)
OP_2 α	$=x$;	m;	$^*m$;	(j)

In the case of a parallel instruction, OP_1 denotes a unary operation determining the new accumulator contents of all activated PEs by a certain transformation of the contents of the register addressed by α as well as the old accumulator contents of the activated PEs; and OP_{l+1} denotes an $(l+1)$-ary operation in the same sense. For the activated PE(j) the operation address m indicates the contents of register (j, m), $^*m$ indicates the contents of register (j, n) if the nonnegative integer n is the contents of register (j, m) at that moment (i.e., indirect operand addressing, in any situation of incorrect programming; e.g., in the case that (j, m) does not have a nonnegative integer contents at that moment, an interrupt of the programmed system is assumed), and the operand $: i_1, i_2, \ldots, i_l$ for $l \geqq 1$ indicates the contents of the accumulators of those neighbors of the activated PEs that are encoded by

$i_1, i_2, \ldots, i_l$ according to the edge coding scheme of the interconnection network. LOAD and STORE have the obvious meanings that the accumulator contents of the activated PEs are replaced by the addressed value, or copied to the addressed registers, respectively. READ and WRITE denote the necessary operations for communication with the outside world where the source and the destination of the data in the "outside world" remain unspecified (certain places within a computing environment not belonging to the given SIMD system itself). In the case of a serial instruction, the unary operation OP_1 and the binary operation OP_2 produce new CPU accumulator contents by a certain transformation of the addressed values, where in the case of OP_2 the old CPU accumulator contents is used as the operand in the first position. READ, WRITE, LOAD, and STORE have the obvious fixed meanings. The operands $=x, m, {}^*m$, and (j) indicate the data unit x itself, the contents of CPU register m, the contents of CPU register n if register m contains the nonnegative number n at that moment, and the contents of register $(j, 0)$, respectively. Note that with this abstract CPU instruction set data transfer between the CPU and the PEs is possible via the accumulators in serial mode only. Furthermore, for a specialized SIMD model, it is convenient to identify the basic computational power of the PEs and the CPU with that of the RAM as represented by the RAM instruction set [2, Fig. 15], roughly speaking. In this way, an interesting point is provided by the description of how the PEs are able to perform local logical decisions in SIMD mode as we shall explain in Example 1 by equation (1) for a special SIMD model.

Off-line I/O *convention.* For the off-line communication of an SIMD system with the outside world we assume that a special set of input registers of the system is fixed such that all other registers of the system contain value zero at the beginning of any computation (moment $t=0$) as it is assumed for those input registers not actually needed for the placement of input data. Each of the input registers may contain at most one data unit of the input data. Thus, for concrete problem solutions, it is necessary to specify

- what data structure is assumed for the given input data, and
- how the data are placed in the given input register set.

Also, a set of output registers of the system must be fixed. In this sense, for concrete problem solutions it has to be clear

- what is the desired data structure for the output data, and
- how this data structure has to be stored, or computed in the predetermined output register set.

As off-line I/O convention we declare that for a certain L, $1 \leqq L \leqq D_{\text{CPU}}$, the CPU registers $0, 1, \ldots, L-1$ are fixed to be input and output registers, and for any $\text{PE}(j)$, if there exists a certain $m \geqq 0$ such that register (j, m) is fixed to be an input register (output register) then register $(j, 0)$ is an input register (output register) as well. What is true for the register holds for the accumulator, too.

On-line I/O *convention.* For the on-line communication of an SIMD system with the outside world some registers are predetermined to act as input and/or output registers. As on-line I/O convention we adopt the same rules as in the off-line case. But, at the beginning of any on-line computation (moment $t=0$), all registers of the system are assumed to hold value zero. Input data or output data may enter or leave the system at a moment as specified by the CPU program according to READ or WRITE instructions. In any correct program these input (output) instructions have to be addressed to a proper subset of all registers specified as input (output) registers. For the input (output) data it is assumed that there exists a memory facility in the outside world from where (to where) the input (output) data are obtained (given) by the system. Thus, for concrete problem solutions it is necessary to specify

- what data structures are assumed for the input and output data, and
- how these data are partitioned into waves of information such that one wave may enter (leave) the system per input (output) operation as performed according to the CPU program.

The size of these waves of information, i.e., the number of data units forming those waves, may alter during a computation process, and just one data unit, for example by LOAD $= x$, will be considered to be the simplest case of a wave of information.

Uniform cost criterion. For measuring the time complexity of computations, we assume that any (basic) instruction of the SIMD system needs one unit of time for performance on this system.

Definition 1. A model of computation SYS is called a *standard off-line network system* (SYS∈OFF-NET) iff SYS is defined by

- a CPU and a fixed set of indexed PEs, with concrete values for D_{CPU} and D_{PE},
- abstract registers if not otherwise specified, and the standard register enumeration,
- a uniform interconnection network with $0 \leqq N_{\text{IN}} < \infty$,
- the standard PE masking scheme,
- a special interpretation and selection of instructions of the abstract CPU instruction set where

(OFF. 1) no READ and WRITE instructions are contained in the instruction set of SYS,

(OFF. 2) for the CPU all RAM instructions [2, Fig. 1.5] except READ and WRITE are avilable,

(OFF. 3) for $N_{\text{IN}} = p \geqq 1$ at least one instruction of the type [all PE's] OP_{p+1}: $0, ., ..., p-1$ is available, and

(OFF. 4) for any output register $(j, 0)$, i.e., accumulator of PE(j), at least one instruction of the type $\text{OP}_2(j)$ is available, i.e., the CPU may have control of any outputting PE,

- the off-line I/O convention, and
- the uniform cost criterion.

For the defined class OFF-NET we may define subclasses — e.g., OFF-NET$_{\text{P}}$ to be the set of all SYS∈OFF-NET having the branching degree $p = N_{\text{IN}}$, OFF-SQUARE to be the set of all SYS∈OFF-NET having a square network as defined in Table 1, OFF-BINTREE with the same reference of Table 1, OFF-PS$=$ $= \bigcup_{m=1}^{\infty}$ OFF-PS^m, or just OFF-RAM.

Example 1. Let us consider the following special SIMD system EXAMP1∈ ∈OFF-SQUARE. Let $D_{\text{CPU}} = D_{\text{PE}} = \infty$. Additionally to the CPU registers $0, 1, ..., L-1$ for a certain $L \geqq 1$, all the accumulators $(j, k, 0)$, $0 \leqq j < M$ and $0 \leqq k < N$ for some $M, N \geqq 1$, are fixed as input and output registers of EXAMP1. The system possesses the following instruction set:

[mask] ADD α, α for $m, {}^*m,\ :i_1, \ldots, i_l$ for $i_1, \ldots, i_l \in \{0, 1, 2, 3\}$,
[mask] $\mathrm{OP}_l\ \alpha, \alpha$ for $m, {}^*m,\ :i$ for $i \in \{0, 1, 2, 3\}$, $l = 1, 2$,
[mask] LOAD α, α for $m, {}^*m,\ :i$ for $i \in \{0, 1, 2, 3\}$,
[mask] STORE α, α for $m, {}^*m,\ :i_1, \ldots, i_l$ for $i_1, \ldots, i_l \in \{0, 1, 2, 3\}$,
LOAD α, α for $=x, m, {}^*m, (j, k)$,
STORE α, α for $m, {}^*m, (j, k)$,
$\mathrm{OP}_2\ \alpha, \alpha$ for $=x, m, {}^*m, (j, k)$,
JUMP b, JGTZ b, JZERO b, JLTZ b, and HALT.

Here, [mask] represents an arbitrary PE address mask, OP_1 is ABS (absolute value) or SIGN (signum function), OP_2 is ADD, SUB, MULT, or DIV, for the tuples (j, k) with $0 \leqq j < M$ and $0 \leqq k < N$.

To give a short illustration of the computing power of EXAMP1 let us consider the computation of the *parallel Roberts gradient* (cp. [9] for its importance to digital image processing), where the input image $A = (a_{jk})$ of size $M \times N$ is assumed to be stored in the PE input registers (a_{jk} in register $(j, k, 0)$) at the beginning of the computation. At the end of the computation, value $\max\{|a_{jk} - a_{j+1,k+1}|, |a_{j+1,k} - a_{j,k+1}|\}$ has to be present in register $(j, k, 0)$.

By performing the following sequence pf parallel instructions,

1. [all PEs] STORE 1
2. [all PEs] LOAD :2
3. [all PEs] STORE 2
4. [all PEs] LOAD :1
5. [all PEs] SUB 1
6. [all PEs] ABS 0
7. [all PEs] STORE 3
8. [all PEs] LOAD 1
9. [all PEs] LOAD :1
10. [all PEs] SUB 2
11. [all PEs] ABS 0
12. [all PEs] STORE 4

all registers $(j, k, 3)$ contain value $|a_{jk} - a_{j+1,k+1}|$, and all registers $(j, k, 4)$ contain value $|a_{j+1,k} - a_{j,k+1}|$, for $0 \leqq j < M$ and $0 \leqq k < N$. These values may be considered as two $M \times N$ matrices B and C. For $\max(B, C) = (\max\{b_{jk}, c_{jk}\})$ we have

$$\max(B, C) = B \times \mathrm{sign}\,(B - C) + C \times \mathrm{sign}\,(C - B) + B - B \times \mathrm{sign}\,|B - C|, \qquad (1)$$

where $\times$ means the parallel MULT operation (cross product of two matrices), and sign the parallel SIGN operation. Using this formula, the parallel Roberts gradient may be computed on the defined special OFF-SQUARE system within time 29 or less, independent of the values of M and N, as the reader may check easily. Note that formula (1) describes a way in which the PEs are able to perform local logical decisions in SIMD mode.

Example 2. By some easily described modifications, the system EXAMP1 may be altered dramatically. Replace the square network by $\mathrm{LRUD2I}^m$, for $m < \max\{\log_2 M, \log_2 N\}$, let $W_{\mathrm{PE}} = 1$, and replace the parallel operations ADD, OP_1 and OP_2 by logical operations AND, NOT, and OR, respectively. What results is a special OFF-$\mathrm{LRUD2I}^m$ system EXAMP2 which essentially coincides with the PBS (paralleles Binärbildverarbeitungssystem). The computational power of the PBS was extensively studied in [4].

Definition 2. A model of computation SYS is called a *standard on-line network system* (SYS∈ON-NET) iff SYS is defined by

- a CPU and a fixed set of indexed PEs, with concrete values for D_{CPU} and D_{PE},
- abstract registers if not otherwise specified, and the standard register enumeration,
- a uniform interconnection network with $0 \leqq N_{\text{IN}} < \infty$,
- the standard PE masking scheme,
- a special interpretation and selection of instructions of the abstract CPU instruction set where, for $N_{\text{IN}} \geqq 2$, an integer tuple (p, q) may be denoted to be the *characteristic* of SYS in the following sense:

(ON. 1) $P = N_{\text{IN}}$ and $1 \leqq q < p$,

(ON. 2) a proper subset $\{i_1, i_2, \ldots, i_q\}$ of all directions $\{0, 1, \ldots, p-1\}$ is specified,

(ON. 3) at least one instruction of the type [all PE's] $\text{OP}_{q+1} : i_1, i_2, \ldots, i_q$ is avaible,

(ON. 4) for any of the instructions [mask] LOAD : j or [mask] $\text{OP}_{k(+1)} : j_1, j_2, \ldots, j_k$, $k \geqq 1$, it follows that $j, j_1, j_2, \ldots, j_k \in \{i_1, i_2, \ldots, i_q\}$,

(ON. 5) for any of the instructions [mask] STORE : $j_1, j_2, \ldots, j_k$, $k \geqq 1$, it follows that $j_1, j_2, \ldots, j_k \in \{0, 1, \ldots, p-1\} - \{i_1, i_2, \ldots, i_q\}$, i.e., the result sof consecutive parallel operations may be shifted through the system in directions $\{0, 1, \ldots, p-1\} - \{i_1, i_2, \ldots, i_q\}$ only, and, furthermore

(ON. 6) for the CPU all RAM instructions are avilable including READ and WRITE,

(ON.7) for any output register $(j, 0)$, at least one instruction of the type $\text{OP}_2(j)$ is available,

- the on-line I/O convention, and
- the uniform cost criterion.

For the defined class ON-NET we may define subclasses — e.g., $\text{ON-NET}_{p,q}$ to be the set of all ON-NET systems with characteristic (p, q), ON-LR2I^m to be the set of all SYS$\in$ON-NET having a left-right 2^i network as defined in Table 1, ON-ILLIAC^m with the same reference to Table 1, $\text{ON-PM2I} = \bigcup_{m=1}^{\infty} \text{ON-PM2I}^m$, or just ON-RAM.

Any infinite network class OFF-LINEAR or ON-DIAGONAL may be considered as an abstraction of a finite network system, or as the union of classes of finite network systems in the following way.

Definition 3. Let OFF-IN be the set of all OFF-NET systems which are defined by a special infinite network IN, e.g., IN=LINEAR or $\text{IN} = \text{LRUD2I}^m$. A model of computation SYS is called a *finite OFF-IN system* (SYS$\in$FIN-OFF-IN) iff there exists a system $\text{SYS}_0 \in$OFF-IN such that SYS may be obtained as a restriction of SYS_0 in the following sense:

Let $\mathbf{ind}_0$ and D^0_{PE} be the PE index set and the PE memory depth for SYS_0, respectively. A finite cut-off of the PE register set of SYS_0 is defined by a certain finite subset **ind** of $\mathbf{ind}_0$ and a (possibly infinite) memory depth $D_{\text{PE}} \leqq D^0_{\text{PE}}$. The work of SYS may be described as follows. All registers in a certain finite cut-off of SYS_0 are available in SYS but all registers not in this finite cut-off will be considered to be dummy registers, i.e., they are assumed to store value zero if addressed as an operand, and to "forget" any value handed over to them; this is the only difference between SYS_0 and SYS.

Analogously the set FIN-ON-IN may be defined.

Example 3. An example of a FIN-ON-BINTREE system may be specified as follows. Let $D_{\mathrm{CPU}}=\infty$ and $D_{\mathrm{PE}}=m\geqq 2$. The finite cut-off of the bintree network is given by **ind**$=\{1,2,\dots,2^m-1\}$. Additionally to the CPU accumulator which acts as an input and output register ($L=1$), the registers $(2^{m-1},0)$, $(2^{m-1}+1,0),\dots,(2^m-1,0)$, i.e., the accumulators of the 2^{m-1} leaf node PEs, are fixed as input registers, and register $(1,0)$, i.e., the accumulator of the top node PE, is fixed as an output register. The system possesses the following instruction set:

[mask] ADD α,α for $m, *m, :1, :2, :1,2$,
[mask] OP$_l$ α,α for $m, *m, :1, :2$ and $l=1,2$,
[mask] LOAD α,α for $m, *m, :1, :2$,
[mask] STORE α,α for $m, *m, :0$,
[subset leaf nodes] READ 0,
[top node] WRITE 0,
LOAD α,α for $=x, m, *m, (1)$,
STORE α,α for $m, *m, (1)$,
OP$_l$ α,α for $=x, m, *m, (1)$, and $l=1,2$,
READ 0,
WRITE α,α for $=x, 0$,
JUMP b, JGTZ b, JZERO b, JLTZ b, HALT.

Here, [mask] represents an arbitrary PE address, OP$_1$ either ABS or SIGN, OP$_2$ one of the operation codes ADD, SUB, MULT, or DIV. Altogether, a FIN-ON-BINTREE system EXAMP3 is defined which may be obtained by a restriction of an infinite ON-BINTREE model where infinite sets of input and output PE registers are available in the infinite origin.

To give a short illustration of the computational power of the system EXAMP3 let us consider the computation of the *arithmetical average* $\frac{1}{N}\sum_{i=0}^{N-1} a_i$, $N=2^{n-1}$ and n odd, *for* M *consecutive waves of information* $(a_0,a_1,\dots,a_{N-1})$ where a_i is fed to the accumulator of the PE$(2^{n-1}+i)$, for $i=0,1,\dots,N-1$. In order of the M consecutive waves of information the arithmetical average have to leave the system via register $(1,0)$.

For initialization of the system, at first the instruction LOAD$=N$, STORE (1), [top node] STORE 1 will be performed in this order. For $M\geqq(n-1)/2$ the following sequence of instructions is executed $(n-1)/2$ times:

[leaf nodes] READ 0,
[all PEs] ADD : 1, 2,
[leaf nodes] LOAD 1,
[all PEs] ADD : 1, 2,

followed by the following sequence of instructions which is executed $M-[(n-1)/2]$ times:

[top node] DIV 1,
[top node] WRITE 0,
[leaf nodes] READ 0,
[all PEs] ADD : 1, 2,
[leaf nodes] LOAD 1,
[all PEs] ADD : 1, 2.

Finally, the following sequence of instructions is executed $(n-3)/2$ times:

[top node] DIV 1,
[top node] WRITE 0,
[all PEs] ADD : 1, 2,
[all PEs] ADD : 1, 2,

followed by the last two instructions [top node] DIV 1 and [top node] WRITE 0. Thus, altogether, the arithmetic averages of $M \geqq (n-1)/2$ consecutive waves of information $(a_0, a_1, \ldots, a_{N-1})$ may be computed within $6M+n$ basic operations of EXAMP3, instead of $O(N \cdot M)$ basic operations in the serial case using a RAM as model for computation.

In conclusion, we point out that SIMD now denotes not a general concept (single-instruction, multiple data) but an exactly defined class of models for computation, namely the union of all system classes given by Definitions 1, 2, and 3.

2. Local, global, and total data flow measures

Let SYS$\in$SIMD; throughout this paper such a special parallel processing system will be used as a standard system for considerations of data transfer restrictions in computing systems. Any computational process performed on such a model SYS may be uniquely specified by a CPU program π and a concrete input situation I characterized by the placement of input values into the set of input registers if off-line mode is used, or by the partition of the input data into consecutive waves of information fed to some of the input registers of the system from the outside world if on-line mode is used.

As suggested by applications to visual perception, the set of input registers of the model SYS may be considered as the *retina* of the system, and any new wave of information to this set of input registers represents a snapshot of the outside world. In this sense, after t steps of a computational process characterized by a program π and an input situation I, for any register r of the system we may mark out a certain receptive field $\text{rec}_\pi^I(r, t)$ containing all the names of those input registers which have had any influence on the contents of register r up to the moment t, where new waves of information to the retina of the system create new names of the input registers, formally represented by $r^{(0)}, r^{(1)}, r^{(2)}, \ldots, r^{(i)}, \ldots$ for register r.

Standard register names. At time $t=0$ of any computational process, each register r in our standard enumeration possesses the name $r^{(0)}$. At $t=0$ let the wave number $WN=0$ also. At time $t+1$ assume that a serial or parallel READ instruction, or an instruction LOAD$=x$, OP$_1=x$, or OP$_2=x$ has to be performed. Then, by this operation we obtain $WN \leftarrow WN+1$ and the new names $r^{(WN)}$ for all registers which were addressed by these instructions. For example, the number $(j, c(j, m))^{(WN)}$ in the case of an instruction [mask] READ $^*m$ for all activated processing elements PE(j), where $c(j, m)$ denotes the actual contents of register (j, m), or the name $0^{(WN)}$ in the case of an instruction OP$_2=x$.

Definition 4. Let SYS$\in$SIMD. Standard register names are assumed. For a program π of SYS, an input situation I of SYS, a register r of SYS, and an arbitrary moment $t \geqq 0$, the *receptive field* $\text{rec}_\pi^I(r, t)$ is recursively defined as follows:

moment $t=0$:

$$\mathrm{rec}_\pi^I(r, 0) = \begin{cases} \{r^{(0)}\} & \text{if input register } r \text{ stores an input value according} \\ & \text{to } I\text{, for off-line mode,} \\ & \text{empty set, otherwise} \end{cases}$$

moment $t+1$, $t \geqq 0$:
At moment $t+1$ a certain instruction has to be applied according to π and I, or the HALT instruction is assumed for this moment.

(i) Depending on this instruction, if it is one of those listed in Table 3, the changes of receptive fields are defined as given in this Table where we omit the indices π and I for simplification of the expressions. In the case of parallel instructions, the mentioned changes are valid for all activated PEs PE (j) where j matches [mask].

Table 3. Changes of receptive fields in step $t+1$

Instructions	Changes of receptive fields
[mask] $OP_1\ m$	$\mathrm{rec}\,((j, 0), t+1) = \mathrm{rec}\,((j, m), t)$
[mask] $OP_1\ {*m}$	$\mathrm{rec}\,((j, 0), t+1) = \mathrm{rec}\,((j, m), t) \cup \mathrm{rec}\,((j, c(j, m)), t)$
[mask] $OP_1\ {:i}$	$\mathrm{rec}\,((j, 0), t+1) = \mathrm{rec}\,((f_i(j), 0), t)$
[mask] $OP_2\ m$	$\mathrm{rec}\,((j, 0), t+1) = \mathrm{rec}\,((j, 0), t) \cup \mathrm{rec}\,((j, m), t)$
[mask] $OP_2\ {*m}$	$\mathrm{rec}\,((j, 0), t+1) = \mathrm{rec}\,((j, 0), t) \cup$ $\cup\,\mathrm{rec}\,((j, m), t) \cup \mathrm{rec}\,((j, c(j, m)), t)$
[mask] $OP_{l+1} : i_1, i_2, \ldots, i_l$	$\mathrm{rec}\,((j, 0), t+1) = \mathrm{rec}\,((j, 0), t) \cup \mathrm{rec}\,((f_{i_1}(j), 0), t) \cup$ $\cup\,\mathrm{rec}\,((f_{i_2}(j), 0), t) \cup \ldots \cup \mathrm{rec}\,((f_{i_l}(j), 0), t)$
[mask] STORE m	$\mathrm{rec}\,((j, m), t+1) = \mathrm{rec}\,((j, 0), t)$
[mask] STORE $*m$	$\mathrm{rec}\,((j, c(j, m), t+1) = \mathrm{rec}\,((j, 0), t) \cup \mathrm{rec}\,((j, m), t)$
[mask] STORE $: i_1, i_2, \ldots, i_l$	$\mathrm{rec}\,((f_{i_1}(j), 0), t+1) = \mathrm{rec}\,((j, 0), t)$, $\mathrm{rec}\,((f_{i_2}(j, 0), t+1) =$ $= \mathrm{rec}\,((j, 0), t)$, $\ldots$, $\mathrm{rec}\,((f_{i_l}(j), 0), t+1) = \mathrm{rec}\,((j, 0), t)$
[mask] READ m	$\mathrm{rec}\,(j, m), t+1) = \{(j, m)^{(WN)}\}$
[mask] READ $*m$	$\mathrm{rec}\,((j, c(j, m), t+1) = \mathrm{rec}\,((j, m), t) \cup \{(j, c(j, m))^{(WN)}\}$
$OP_1 = x$	$\mathrm{rec}\,(0, t+1) = \{0^{(WN)}\}$
$OP_1\ m$	$\mathrm{rec}\,(0, t+1) = \mathrm{rec}\,(m, t)$
$OP_1\ {*m}$	$\mathrm{rec}\,(0, t+1) = \mathrm{rec}\,(m, t) \cup \mathrm{rec}\,(c(m), t)$
$OP_1\ (j)$	$\mathrm{rec}\,(0, t+1) = \mathrm{rec}\,((j, 0), t)$
$OP_2 = x$	$\mathrm{rec}\,(0, t+1) = \mathrm{rec}\,(0, t) \cup \{0^{(WN)}\}$
$OP_2\ m$	$\mathrm{rec}\,(0, t+1) = \mathrm{rec}\,(0, t) \cup \mathrm{rec}\,(\mathrm{m}, t)$
$OP_2\ {*m}$	$\mathrm{rec}\,(0, t+1) = \mathrm{rec}\,(0, t) \cup \mathrm{rec}\,(m, t) \cup \mathrm{rec}\,(c(m), t)$
$OP_2\ (j)$	$\mathrm{rec}\,(0, t+1) = \mathrm{rec}\,(0, t) \cup \mathrm{rec}\,((j, 0), t)$
STORE m	$\mathrm{rec}\,(m, t+1) = \mathrm{rec}\,(0, t)$
STORE $*m$	$\mathrm{rec}\,(c(m), t+1) = \mathrm{rec}\,(0, t) \cup \mathrm{rec}\,(m, t)$
STORE (j)	$\mathrm{rec}\,((j, 0), t+1) = \mathrm{rec}\,(0, t))$
READ m	$\mathrm{rec}\,(m; t+1) = \{m^{(WN)}\}$
READ $*m$	$\mathrm{rec}\,(c(m), t+1) = \mathrm{rec}\,(m, t) \cup \{c(m)^{(WN)}\}$

(ii) For the parallel or serial LOAD instructions the changes of receptive fields are the same as for the corresponding OP_1 instructions.

(iii) In the case of a WRITE, JUMP, or HALT instruction no changes of receptive fields appear.

(iv) In the case of a JGTZ, JZERO, or JLTZ instruction no changes of receptive fields appear in step $t+1$, but the set $\mathrm{rec}\,(0, t)$ will be added at moment $t' \geqq t+2$ to any receptive field that alters at moment t' according to (i) or (ii), if at moment t' an instruction has to be performed covered by cases (i) and (ii). For example, the instruction [mask] OP_2 m, at moment $t' \geqq t+2$, will produce the changes $\mathrm{rec}\,((j, 0), t') = \mathrm{rec}\,((j, 0), t'-1) \cup \mathrm{rec}\,((j, m), t'-1) \cup \mathrm{rec}\,(0, t)$ for all activated PEs.

For illustration of this definition, consider the special OFF-SQUARE system as defined in Example 1. Let I be any concrete input situation for computing the parallel Roberts gradient and let π be the sequence of the 12 parallel instructions as given there. At moment $t=0$ we have $\mathrm{rec}\,((j, k, 0), 0) = \{(j, k, 0)^{(0)}\}$, for $0 \leqq j < M$ and $0 \leqq k < N$, and for any other register r of the system EXAMP 1, $\mathrm{rec}\,(r, 0)$ is the empty set. After performing the 12 instructions of π the reception fields of maximal cardinality 2 belong to the registers $(j, k, 0)$, $(j, k, 3)$ and $(j, k, 4)$, for $0 \leqq j \leqq M-2$ and $0 \leqq k \leqq N-2$, where, e.g., $\mathrm{rec}\,((j, k, 0), 12) = \{(j+1, k, 0)^{(0)}, (j, k+1, 0)^{(0)}\}$. For the system defined in Example 3, and the program and the input situation as described there, after performing the $6M+n$ instructions the receptive field of maximal cardinality $NM+1$ belongs to the register $(1, 0)$, i.e., to the accumulator of the top node PE.

Definition 5. Let SYS$\in$SIMD. For a set R of registers of SYS and a moment $t \geqq 0$ define the *local data transfer function* λ_{SYS} by

$$\lambda_{\mathrm{SYS}}(R, t) = \max_{\pi} \max_{I} \max_{r \in R} \mathrm{card}\,(\mathrm{rec}^I(r, t)),$$

the *global data transfer function* γ_{SYS} by

$$\gamma_{\mathrm{SYS}}(R, t) = \max_{\pi} \max_{I} \mathrm{card}\,\Big(\bigcup_{r \in R} \mathrm{rec}^I_\pi(r, t)\Big),$$

the *total data transfer function* τ_{SYS} by

$$\tau_{\mathrm{SYS}}(R, t) = \max_{\pi} \max_{I} \sum_{r \in R} \mathrm{card}\,(\mathrm{rec}^I_\pi(r, t)).$$

By this definition, it follows immediately that the functions $\lambda_{\mathrm{SYS}}, \gamma_{\mathrm{SYS}}$ and τ_{SYS} are monotonically increasing for any set R of registers of SYS and increasing values of t. Furthermore,

$$\lambda_{\mathrm{SYS}}(R, t) \leqq \gamma_{\mathrm{SYS}}(R, t) \leqq \tau_{\mathrm{SYS}}(R, t) \tag{2}$$

for all models SYS$\in$SIMD, sets R of registers and moments $t \geqq 0$. Also note that for any model SYS, if within t steps of an arbitrary program π for SYS starting with an arbitrary input situation I for SYS at most $\omega_{\mathrm{SYS}}(t)$ input data may be fed to the system, then

$$\gamma_{\mathrm{SYS}}(R, t) \leqq \omega_{\mathrm{SYS}}(t), \quad \text{and} \tag{3.1}$$

$$\tau_{\mathrm{SYS}}(R, t) \leqq \lambda_{\mathrm{SYS}}(R, t) \cdot \mathrm{card}\,(R), \tag{3.2}$$

for any set R of registers of SYS and $t \geqq 0$.

Example 4. In Section 4 we shall characterize the way to use these data transfer functions for obtaining lower time bounds for concrete computational problems. For serial data processing we shall apply the system $\mathrm{RAM_L}$, cp. [2, Fig. 1.5], as model for computation, where $R_{\mathrm{L}}=\{0, 1, 2, \ldots, L-1\}$, $L \geqq 1$, is assumed to be the set of all input/output registers of such a machine ($D_{\mathrm{CPU}}=\infty$, $N_{\mathrm{PE}}=0$, $W_{\mathrm{CPU}}=\infty$). For $t \geqq 0$, we have $\omega_{\mathrm{OFF-RAM_L}}(t)=L+t$ and $\omega_{\mathrm{ON-RAM_L}}(t)=t$. For $\text{OFF-RAM}= = \bigcup_{L=1}^{\infty} \text{OFF-RAM}_{\mathrm{L}}$, note that $\omega_{\mathrm{OFF-RAM}}(t)=\max_{L} \omega_{\mathrm{OFF-RAM_L}}(t)$ is not defined. Furthermore, we have

$$\lambda_{\mathrm{OFF-RAM_L}}(R_L, t) = \begin{cases} 2t+1 & \text{for} \quad 0 \leqq t \leqq \lfloor (L-1)/2 \rfloor \\ \lfloor (L+1)/2 \rfloor + t, & \text{otherwise,} \end{cases} \tag{4.1}$$

$$\gamma_{\mathrm{OFF-RAM_L}}(R_L, t) = L+t, \quad \text{and} \tag{4.2}$$

$$\tau_{\mathrm{OFF-RAM_L}}(R_L, t) = L(t - \lfloor L/2 \rfloor + 1) \quad \text{for} \quad t \geqq \lfloor L/2 \rfloor, \tag{4.3}$$

in the case of using the $\mathrm{RAM_L}$ in off-line mode, and

$$\lambda_{\mathrm{ON-RAM_L}}(R_L, t) = \gamma_{\mathrm{ON-RAM_L}}(R_L, t) = t,$$

$$\tau_{\mathrm{ON-RAM_L}}(R_L, t) = \begin{cases} t(t+l)/2 & \text{for} \quad t \leqq L \\ L\big(t-(L/2)+1/2\big) & \text{for} \quad t \geqq L, \end{cases} \tag{4.5}$$

in the case of using the $\mathrm{RAM_L}$ in on-line mode. The maximal data flow for obtaining equation (4.1) is possible by indirect addressing $\mathrm{OP_2}\ {*m}$, followed by $\mathrm{OP_2}{=}x$ operations. For (4.3), the same sequence of operations is extended by $L-1$ instructions STORE m. For (4.4), t operations of the type $\mathrm{OP_2}{=}x$ may be considered. For small t the exact derivation of the function $\tau_{\mathrm{OFF-RAM_L}}$ represents a sophisticated problem already, for this quite simple model of serial computation.

Example 5. For further illustration of the concrete derivation of these data transfer functions, let us consider both systems EXAMP1 and EXAMP3 as defined above.

For the system EXAMP1, first we see that $\omega_{\mathrm{EXAMP1}}(t)=MN+L+t$, for $t \geqq 0$. Let $R_{M,N}$ be the set $\{(j, k, 0) : 0 \leqq j < M \text{ and } 0 \leqq k < N\}$ of all PE input/iutput registers of the system. By using t operations of the type

[all PE's] ADD :0, 1, 2, 3

we obtain the maximal local and total data transfer within the field of PE accumulators, where

$$\lambda_{\mathrm{EXAMP1}}(R_{M,N}, t) = 2t^2+2t+1, \tag{5.1}$$

$$(2t^2+2t+1)MN - \left(\frac{t+1}{3} - (t+1)^2 + \frac{2(t+1)^3}{3}\right)(M+N) \leqq$$

$$\leqq \tau_{\mathrm{EXAMP1}}(R_{M,N}, t) \leqq (2t^2+2t+1)MN, \tag{5.2}$$

for $2t+1 \leqq \min\{M, N\}$, by elementary combinatorial considerations and (3.2). For $t \geqq t_0 = \lfloor M/2 \rfloor \cdot \lfloor N/2 \rfloor$ we have

$$MN + (t-t_0) \leqq \lambda_{\mathrm{EXAMP1}}(R_{M,N}, t) \leqq MN+L+t. \tag{4.3}$$

For $t \geqq t_0 = M+N-2$ we can easily see that

$$M^2N^2+(t-t_0) \leqq \tau_{\text{EXAMP1}}(R_{M,N}, t) \leqq MN(MN+L+t). \tag{5.4}$$

Finally, for the case of global data transfer we obtain

$$\gamma_{\text{EXAMP1}}(R_{M,N}, t) = \begin{cases} MN & \text{for } \ t=0 \\ MN+2t+1 & \text{for } \ 2t+1 \leqq L \ \text{ and } \ t>0 \\ MN+\lfloor (L-1)/2 \rfloor + t & \text{for } \ 2t+1 > L \end{cases} \tag{5.5}$$

where, for $2t+1 \leqq L$, the maximal global data transfer is possible by t operations of the type ADD $*m_t$ and one operation STORE (j, k), e.g.

For the system EXAMP3, at first we have $\omega_{\text{EXAMP3}}(t) = t \cdot N$, for $N = 2^{n-1}$ and $t \geqq 0$ by using t operations of the type

[leaf nodes] READ 0.

Let $R_0 = \{0, (1, 0)\}$ be the set of the two distinguished output registers of this syste EXAMP3. By using the instruction pair

[leaf nodes] READ 0,
[all PEs] ADD :1, 2

repeated $(m-1)$ times, $m \geqq 1$; the single instruction

[leaf nodes] READ 0

again; and finally $(n-1)$ instructions

[all PEs] ADD : 1, 2,

we obtain the maximal local data transfer for register $(1, 0)$ in any case $t \geqq m$. We have

$$\lambda_{\text{EXAMP3}}(R_0, t) = \begin{cases} 0 & \text{for } \ t=0 \\ 2^{t-1} & \text{for } \ 1 \leqq t \leqq n-1 \\ m \cdot N & \text{for } \ t = n+2m-l, \quad m \geqq 1 \\ & \text{and } \ l=1 \ \text{ or } \ l=2, \end{cases}$$

for all $t \geqq 0$. Analogously, for the same set R_0 and $t \geqq 0$

$$\gamma_{\text{EXAMP3}}(R_0, t) = \begin{cases} 0 & \text{for } \ t=0, \\ 2^{t-1} & \text{for } \ 1 \leqq t \leqq n-1, \\ m \cdot N & \text{for } \ t = n+2m-2, \quad m \geqq 1, \\ m \cdot N+1 & \text{for } \ t = n+2m-1, \quad m \geqq 1, \end{cases}$$

$$\tau_{\text{EXAMP3}}(R_0, t) \begin{cases} 0 & \text{for } \ t=0, \\ 2^{t-1} & \text{for } \ 1 \leqq t \leqq n+1, \\ 2m \cdot N & \text{for } \ t = n+2m-1, \quad m \leqq 1. \\ 2m \cdot N+1 & \text{for } \ t = n+2m, m \geqq 1. \end{cases}$$

Of course, the values of $\lambda_{\text{EXAMF3}}, \gamma_{\text{EXAMP3}}$, and τ_{EXAMP3} depend on the choice of the set R_0, and may be quite different for some other sets of registers.

Definition 6. Let CLASS $\subseteqq$ SIMD. The *general data transfer functions* are defined as follows, for such a set CLASS of models of computation, for $t, n \geqq 0$:

$\Lambda_{\text{CLASS}}(t)$ denotes the maximal value of all $\lambda_{\text{SYS}}(R, t)$,

$\Gamma_{\mathrm{CLASS}}(n, t)$ denotes the maximal value of all $\gamma_{\mathrm{SYS}}(R, t)$ with card $(R)=n$, and $T_{\mathrm{CLASS}}(n, t)$ denotes the maximal value of all $\tau_{\mathrm{SYS}}(R, t)$ with card $(R)=n$, where SYS is an arbitrary element of CLASS, and R denotes a set of registers of SYS.

Interesting examples of CLASS are sets like OFF-NET$_p$, ON-NET$_{p,q}$, OFF-SQUARE, OFF-BINTREE, or ON-HEXAGONAL, where these general data transfer functions are fully defined.

Theorem 1. For standard off-line network systems and $2 \leqq p < \infty$ we have

$$\Lambda_{\mathrm{OFF-NET}_p}(t) = \begin{cases} 2t+1 & \text{for } p=2 \\ p\left(\dfrac{(p-1)^t-1}{p-2}\right)+1 & \text{for } p \geqq 3, \end{cases}$$

and

$$\Gamma_{\mathrm{OFF-NET}_p}(n, t) = T_{\mathrm{OFF-NET}_p}(n,t) = n \cdot \Lambda_{\mathrm{OFF-NET}_p}(t), \quad \text{for} \quad n, t \geqq 0.$$

Proof. First, let us consider the local situation. For $p=2$, the maximal transfer of data units is possible by indirect addressing to the CPU accumulator, e.g. For $p \geqq 3$, there exist special OFF-NET$_p$ models SYS$_t$ such that, according to (OFF.3), at any moment $1 \leqq s \leqq t$ the maximal possible number of $p(p-1)^{s-1}$ new names of input registers may enter the receptive field of a certain register r, for $t \geqq 0$. Thus,

$$\lambda_{\mathrm{SYS}_t}(\{r\}, t) = 1 + \sum_{s=0}^{t-1} p(p-1)^s = p\left(\frac{(p-1)^t-1}{p-2}\right)+1.$$

For the total and global situation note that by choosing sufficiently complex SYS$_{n,t}$, for $n, t \geqq 0$, the maximal local situations of data transfer characterized by receptive fields of cardinality $\Lambda_{\mathrm{OFF-NET_P}}(t)$ at moment t may appear in n different registers and time t such that these registers are far enough from one another so that their receptive fields are pairwise disjoint. □

Example 6. By (4.1) and Theorem 1, it follows that $\Lambda_{\mathrm{OFF-RAM}}(t) = \Lambda_{\mathrm{OFF-NET}_2}(t) = 2t+1$, for $t \geqq 0$. Of course, this coincidence is not true in the total and global cases. According to Theorem 1 we have $\Gamma_{\mathrm{OFF-NET}_2}(n, t) = T_{\mathrm{OFF-NET}_2}(n, t) = n(2t+1)$, for $n, t \geqq 0$, but by elementary considerations $\Gamma_{\mathrm{OFF-RAM}}(n, t) = 2t+n$, for $n \geqq 1$ and $T_{\mathrm{OFF-RAM}}(n, t) = 2n(t-n+2)-2$, for $t \geqq n \geqq 2$.

In Table 4 the general local data transfer functions are collected for some classes of off-line systems as defined in Section 1. For these classes, the functions $\Lambda_{\mathrm{OFF-NET}_p}$ as given in Theorem 1 act as upper bounds, where the proper value of p has to be specified. The classes OFF-LINEAR, OFF-PS, OFF-BINTREE and OFF-QUADTREE represent examples for the maximal transfer situations as characterized by Theorem 1, for $p=2, 3, 5$, respectively.

Some remarks about Table 4 and about the other networks which were defined in Table 1.

1. For the bintree, triangle and quadtree network note that the maximal receptive fields may be obtained for central nodes of these tree structures only, and not at the top node. The maximal possible cardinalities of receptive fields of top node accumulators are given for illustration of this fact.

Table 4. General local data transfer functions for offline systems

CLASS	P	$\Lambda_{\text{OFF—CLASS}}(t)$	$t=4$	$t=8$
LINEAR	2	$2t+1$	9	17
HEXAGONAL	3	$\frac{3}{2}t^2+\frac{3}{2}t+1$	31	109
SQUARE or ILLIAC	4	$2t^2+3t+1$	41	145
TRIAGONAL	6	$3t^2+3t+1$	61	215
DIAGONAL	8	$4t^2+4t+1$	81	289
PS	3	$3\cdot 2^t-2$	46	766
BINTREE	3	$3\cdot 2^t-2$	46	766
top node		$2^{t+1}-1$	31	511
TRIANGLE	5	$3\cdot 2^{t+1}+t^2-2t-5$	99	1,579
top node		$2^{t+1}-1$	31	511
QUADTREE	5	$(5\cdot 4^t-2)/3$	426	109, 226
top node		$(4^{t+1}-1)/3$	341	87, 381

2. For all examples of CLASS given in Table 4, we have $\Gamma_{\text{OFF—CLASS}}(n, t)=T_{\text{OFF—CLASS}}(n, t)=n\cdot\Lambda_{\text{OFF—CLASS}}(t)$, for $n, t\geqq 0$.

3. The hexagonal, square, triagonal, and diagonal networks are special examples of infinite graphs of constant degree p such that the general local data transfer function is equal to $\frac{p}{2}t^2+\frac{p}{2}t+1$. Such networks correspond to usual digital metrics for the orthogonal grid in a natural way, e.g., the metrics d_4 or d_8 as used in digital image processing, cp. [9], to the square or diagonal network, respectively.

4. For the networks CUBE^m, PM2I^m, WPM2I^m, LR2I^m, or LRUD2I^m, the derivation of the three general data transfer functions represents a very sophisticated problem. Of course, the values of these functions depend on the value of m, and the consideration of classes like

$$\text{CUBE} = \bigcup_{m\geqq 2}\text{CUBE}^m$$

would lead to undefined general data transfer functions. In [4] the general local data transfer functions were analyzed for some concrete SIMD systems similar to FIN-OFF-LR2I^m or FIN-OFF-LRUD2I^m systems like EXAMP2 which was defined above. But, for the present paper, we recommend data transfer analysis for specialized (finite) SIMD systems to the interested reader, and are satisfied with some hints:

CUBE^m: For this system, the exact derivation of the local transfer function should be a solvable task. We have

$$\Lambda_{\text{OFF—CUBE}^m}(t)\begin{cases} = \sum\limits_{i=0}^{t}\binom{m}{i} & \text{for} \quad t<m \\ \geqq 2^m & \text{for} \quad t=m \\ \geqq 2^{m+1}(t-m) & \text{for} \quad t>m. \end{cases}$$

For example, we have $\Lambda_{\text{OFF—CUBE}}256\,(4)=177{,}589{,}057$ and $\Lambda_{\text{OFF—CUBE}}256(8)$ is about $4\cdot 10^{14}$.

PM2I^m: For this, as for the other "power-of-two systems", the analysis of data flow represents quite a hard problem, cp. [4]. But, to give the reader some feeling about the complexity of the data transfer functions for these systems, some values will be collected:

$$\Lambda_{\text{OFF}-\text{PM2I}^m}(t)\begin{cases} = 1 & \text{for} \quad t = 0 \\ = 2 & \text{for} \quad t = 1 \\ = 2(m-1)(m-2)+4 & \text{for} \quad t = 2 \\ \vdots & \qquad\;\; \vdots \\ \geqq 2^m & \text{for} \quad t = \lceil m/2 \rceil \\ \geqq 2^{m+1}(t-\lceil m/2 \rceil) & \text{for} \quad t > \lceil m/2 \rceil. \end{cases}$$

Note that exponential increase changes to linear increase at $t = \lceil m/2 \rceil$.

WPM2I^m: It may be that this is the most complicated situation of any network; we have

$$\Lambda_{\text{OFF}-\text{WPM2I}^m}(t)\begin{cases} = 1 & \text{for} \quad t = 0 \\ = 2 & \text{for} \quad t = 1 \\ \vdots & \qquad\;\; \vdots \\ \geqq 2^m & \text{for} \quad t = \lceil m/2 \rceil \\ \geqq 2^{m+1}(t-\lceil m/2 \rceil) & \text{for} \quad t \geqq \lceil m/2 \rceil. \end{cases}$$

This great difficulty in analyzing data paths should be a hint to the limited practical importance of this network.

LR2I^m: For brevity we shall use the function $\sigma(i) = \sum\limits_{j=1}^{i} j^2 = \frac{1}{6}(i+1) - \frac{1}{2}(i+1)^2 + \frac{1}{3}\,1(i+1)^3$. We found the following interesting values:

$$\Lambda_{\text{OFF}-\text{LR2I}^m}(t) = \begin{cases} 1 & \text{for} \quad t = 0 \\ 2m+1 & \text{for} \quad t = 1 \\ 2(m-2)^2+4m+1 & \text{for} \quad t = 2 \\ 1+6m+4(m-2)^2+2\cdot\sigma(m-4) & \text{for} \quad t = 3 \\ 1+8m+6(m-2)^2+4\cdot\sigma(m-4)+ & \\ \qquad +4\cdot\sum\limits_{i=1}^{m-6}\sigma(i) & \text{for} \quad t = 4 \\ 1+10m+8(m-2)^2+6\cdot\sigma(m-4)+ & \\ \qquad +8\cdot\sum\limits_{i=1}^{m-6}\sigma(i)+ & \text{for} \quad t = 5 \\ \qquad +8\sum\limits_{i=1}^{m-8}\sum\limits_{j=1}^{i}\sigma(j) & \\ \qquad\qquad \vdots & \qquad\;\; \vdots \\ 2^m\cdot t - c_m & \text{for} \quad t \geqq \lfloor (m-1)/2 \rfloor \end{cases}$$

The contents c_m depend on the value of m only, for example $c_2=-1$, $c_3=1$, $c_4=7$, $c_5=25$, $c_6=71$, $c_7=185$, $c_8=455$, $c_9=1081$, and $c_{10}=2503$. Because the LR2I^m is an infinite network $\Gamma_{\text{OFF—LR2I}^m}(n,t)=T_{\text{OFF—LR2I}^m}(n,t)=n\cdot\Lambda_{\text{OFF—LR2I}^m}(t)$, for $n, t\geqq 0$.

LRUD2I^m: Of course, we have $\Lambda_{\text{OFF—LRUD2I}^m}(t)\geqq 2\cdot\Lambda_{\text{OFF—LR2I}^m}(t)-1$, for $t\geqq 0$, and, because LRUD2I^m is an infinite network we have $\Gamma_{\text{OFF—LRUD2I}^m}(n,t)=T_{\text{OFF—LRUD2I}^m}(n,t)=n\cdot\Lambda_{\text{OFF—LRUD2I}^m}(t)$, for $n, t\geqq 0$.

Theorem 2. For standard on-line network systems and $2\leqq p<\infty$, $1\leqq q\leqq p-1$,

$$\Lambda_{\text{ON—NET}_{p,q}}(t)=\begin{cases}0 & \text{for } t=0,\\ 2t-1 & \text{for } t\geqq 1 \text{ and } q=1,\\ (q^t-1)/(q-1) & \text{for } t\geqq 1 \text{ and } q\geqq 2,\end{cases}$$

and $\Gamma_{\text{ON—NET}_{p,q}}(n,t)=T_{\text{ON—NET}_{p,q}}(n,t)=n\cdot\Lambda_{\text{ON—NET}_{p,q}}(t)$, for $n, t\geqq 0$.

Proof. Consider the local data transfer situation first. At $t=1$ assume that a sufficiently large set of input registers obtain input data in parallel by a READ instruction. Then $(q-1)/(q-1)=2t-1=1$ for $q\geqq 2$, or $t=1$. For $q=1$, the maximal local transfer situation, i.e., the maximal transfer of data units to a given register, is possible by indirect addressing. Thus, $\Lambda_{\text{ON—NET}p,1}(t)=2t-1$ for $t\geqq 1$. For $q\geqq 2$, according to (ON.3) it follows that

$$\Lambda_{\text{ON—NET}_{p,q}}(t)=\sum_{i=0}^{t-1}q^i=(q^t-1)/(q-1),$$

where these maximal cardinalities of receptive fields may be obtained in certain PE accumulators. For given $n, t\geqq 0$, by choosing a sufficiently large field of PEs obtaining input data in their accumulators at the first instruction $(i=1)$, n receptive fields of maximal cardinality $\Lambda_{\text{ON—NET}p,q}(t)$ may be pairwise disjoint. □

Example 7. By (4.4) we know that $\Lambda_{\text{ON—RAM}}(t)=\Gamma_{\text{ON—RAM}}(n,t)=t$, for $t\geqq 0$ and $n\geqq 1$, and thus $\Lambda_{\text{ON—RAM}}(t)<\Lambda_{\text{ON—NET}_{p,1}}(t)$ as well as $\Gamma_{\text{ON—RAM}}(n,t)<\Gamma_{\text{ON—NET}_{p,1}}(n,t)$ for $t\geqq 2$ and $n\geqq 1$. Furthermore, $T_{\text{ON—RAM}}(n,t)=n\left(t-\frac{n}{2}+\frac{1}{2}\right)$, for $t\geqq n\geqq 1$, and thus $T_{\text{ON—RAM}}(n,t)<T_{\text{ON—NET}_{p,1}}(n,t)$ for $t\geqq n\geqq 2$.

In table 5 for classes of on-line systems mentioned in Section 1 some results on the analysis of general local data transfer functions are collected. For these classes the functions given in Theorem 2 act as upper bounds where the proper values of p and q have to be correlated. By ON-IN$_{\{i_1,i_2,\dots,i_q\}}$ we denote a special ON-IN system with fixed set $\{i_1, i_2, \dots, i_q\}$ according to (ON.2). The classes ON-LINEAR$_{\{0\}}$, ON-BINTREE$_{\{1,2\}}$, and ON-QUADTREE$_{\{1,2,3,4\}}$ represent examples for maximal transfer situations as characterized by Theorem 2.

Some remarks about Table 5 and about the other networks which were defined in Table 1:

1. For all examples of CLASS in Table 5 we have $\Gamma_{\text{ON—CLASS}}(n,t)=T_{\text{ON—CLASS}}(n,t)=n\cdot\Lambda_{\text{ON—CLASS}}(t)$, for $n, t\geqq 0$.

Table 5. General local data transfer functions for on-line systems

CLASS	p	$\{i_1, i_2, \ldots, i_q\}$	$\Lambda_{\text{ON--CLASS}}(t)$	$t=4$	$t=8$
LINEAR	2	$\{0\}$	$2t-1$	7	15
HEXAGONAL	3	$\{0, 1\}$ $\{0\}$	$t(t+1)/2$ $2t-1$	10 7	36 15
SQUARE or ILLIAC	4	$\{0, 1, 2\}$ $\{0, 2\}$ $\{0, 1\}$, $\{0\}$	t^2 $t(t+1)/2$ $2t-1$	16 10 7	64 36 15
TRIAGONAL	6	$\{0, 1, 2, 3, 4\}$ $\{0, 2, 3, 4\}$ $\{0, 2, 4\}$	$\frac{5}{2}t^2-\frac{5}{2}t+1$ $\frac{3}{2}t^2-\frac{1}{2}t$ t^2	31 22 16	121 92 64
DIAGONAL	8	$\{0, 1, 2, 3, 4, 6, 7\}$	$\frac{7}{2}t^2-\frac{7}{2}t+1$	43	197
BINTREE	3	$\{1, 2\}$ $\{0, 1\}$	2^t-1 $t(t+1)/2$	15 10	255 36
TRIANGLE	5	$\{1, 2, 3, 4\}$	2^t-1	15	255
QUADTREE	5	$\{1, 2, 3, 4\}$	$(4^t-1)/3$	85	21,845
PS	3	$\{0, 1\}$	$([(1+\sqrt{5})^{t+3}-(1-\sqrt{5})^{t+3}]/\sqrt{5}\cdot 2^{t+3})-2$	11	87

2. The class ON-PS$_{\{0,1\}}$ denotes special SIMD systems using the PS network in its original [10] meaning. Let $f_0=1, f_1=1, f_2=2, \ldots, f_{n+2}=f_n+f_{n+1}, \ldots,$ where

$$f_n=[(1+\sqrt{5})^{n+1}-(1-\sqrt{5})^{n+1}/\sqrt{5}\cdot 2^{n+1}$$

denotes the nth Fibonacci number, $n\geqq 0$. We have $\Lambda_{\text{ON--PS}_{\{0,1\}}}(t)=\sum_{n=1}^{t} f_n=f_{n+2}-2$, for $t\geqq 0$; cp. [3] for a similar result.

3. For the bintree, triangle, and quadtree network note that the maximal receptive fields may be obtained for the top node accumulator, for $\{i_1, i_2, \ldots, i_q\}$ equal to $\{1, 2\}$, $\{1, 2, 3, 4\}$, $\{1, 2, 3, 4\}$, respectively.

4. The analysis of the general data transfer functions for classes ON-CUBE^m, ON-PM2I^m, ON-WPM2I^m, ON-LR2I^m, and ON-LRUD2I^m will not be considered in the present paper.

3. Local, global, and total data dependence measures

For parallel processing systems, the optimal time for the solution of a computational problem depends upon the data transfer abilities of the given system as well as on the principal possibilities of parallelization of a solution process for a given problem. The first may be characterized by the data transfer functions Λ_{SYS}, Γ_{SYS}, T_{SYS} by a general system analysis as considered in Section 2. The second property, however, requires individual consideration of the given computational problem.

For example, consider the multiplication of two $N\times N$ real matrices $A\cdot B=C$. For a given system SYS assume that all N^2 elements of matrix C have to be computed in N^2 different output registers represented by the set R_{OUT}. Let $r\in R_{\mathrm{OUT}}$, $R_0\subseteqq R_{\mathrm{OUT}}$, and R_1 be the set of N distinctive registers for outputing the N diagonal elements of C. Then it follows that $\lambda_{\mathrm{SYS}}(r, t^*)\geqq 2N$, $\gamma_{\mathrm{SYS}}(R_1, t^*)\geqq$ $\geqq 2N^2$ and $\tau_{\mathrm{SYS}}(R_0, t^*)\geqq 2N\cdot\mathrm{card}\,(R_0)$ if the product $A\cdot B$ is to be computed on SYS within time t^*. Thus, if the functions Λ_{SYS}, Γ_{SYS} or T_{SYS} are known, lower time bounds are derivable from these inequalities for the solution time t^* immediately, where the maximal lower time bound from the three possible values is taken as the result. For example, according to our considerations in Section 2 for the system EXAMP1 we have $t^*\geqq\sqrt{N}-1$ under the assumption that $M=2N$. But note that a better lower time bound for this system and the matrix multiplication problem may be obtained by more specialized considerations as demonstrated by Gentleman [3, Theorem 1]. Because each data unit transfer from a certain register r_1 to a certain register r_2 of the system EXAMP1 may be performed in the reverse direction, from r_2 to r_1, in the same time, the proof of Theorem 1 in [3] matches the situation given by the system EXAMP1, i.e., for $r\in R_{\mathrm{OUT}}$ we have $\lambda_{\mathrm{EXAMP1}}\ (r, 2t^*)\geqq N^2$, and thus $t^*\geqq\frac{1}{4}(2N^2-1)^{1/2}-\frac{1}{4}$.

For a general approach to the derivation of lower time bounds for parallel processing systems we shall use the quantitative description of data dependencies of the desired output data in relation to the input data specification, for computational problems which may be identified with special functions as described later on.

Definition 7. Let $n, m\geqq 1$. Let f be an n-ary function defined on a certain set *domain* (f) of n-tuples of real numbers, and into the set of m-tuples of real numbers. For an n-tuple $(x_1, x_2, \dots, x_n)\in$*domain* (f), define

$$\mathrm{sub}_i\,(x_1, x_2, \dots, x_n)=\{j:1\leqq j\leqq n\,\&(\vee x'\neq x_j)(x_1, x_2, \dots, x_{j-1}, x', x_{j+1}, \dots, x_n)\in$$

$$\in \textit{domain}\,(f)\ \&\ \mathrm{proj}_i\,(f(x_1, x_2, \dots, x_n))\neq \mathrm{proj}_i\,(f(x_1, x_2, \dots, x_{j-1}, x', x_{j+1}, \dots, x_n))\}$$

to be the set of all positions j such that changes in the jth component of $(x_1, x_2, \dots, x_n)$ have an effect on the projection $\mathrm{proj}_i\,f$, for $1\leqq i\leqq m$. Then, define

$$\lambda_f=\max_{(x_1, x_2, \dots, x_n)}\ \max_{1\leqq i\leqq m}\ \mathrm{card}\,(\mathrm{sub}_i\,(x_1, x_2, \dots, x_n)),$$

$$\gamma_f=\max_{(x_1, x_2, \dots, x_n)}\ \mathrm{card}\left(\bigcup_{i=1}^{m}\mathrm{sub}_i\,(x_1, x_2, \dots, x_n\right),$$

and

$$\tau_f = \max_{(x_1, x_2, \ldots, x_n)} \sum_{i=1}^{m} \operatorname{card}(\operatorname{sub}_i(x_1, x_2, \ldots, x_n)).$$

The function f is called *locally d-dependent* iff $d \leqq \lambda_f$, *globally d-dependent* iff $d \leqq \gamma_f$, and *totally d-dependent* iff $d \leqq \tau_f$, for an integer $d \geqq 0$.

By this definition, for arbitrary functions f defined on n-tuples of real numbers and into the set of m-tuples of real numbers, it follows immediately that $\lambda_f = \gamma_f = \tau_f$ if $m = 1$, and for $m \geqq 1$

$$\lambda_f \leqq \gamma_f \leqq \tau_f, \tag{7.1}$$

$$\gamma_f \leqq n, \tag{7.2}$$

and

$$\tau_f \leqq m \cdot \lambda_f. \tag{7.3}$$

For example, in the case of the following function f,

$$f(x_1, x_2, x_3, x_4, x_5) = \begin{cases} x_1 + x_2 & \text{if} \quad x_5 = 0 \\ x_3 + x_4 & \text{if} \quad x_5 \neq 0, \end{cases}$$

we have $\operatorname{sub}_1(x_1, x_2, x_3, x_4, 0) = \{1, 2, 5\}$ if $x_1 + x_2 \neq x_3 + x_4$, and $\operatorname{sub}_1(x_1, x_2, x_3, x_4, 0) = \{1, 2\}$ if $x_1 + x_2 = x_3 + x_4$. Because of $\lambda_f = \gamma_f = \tau_f = 3$, this function is local, global, or total 1-, 2-, and 3-dependent, but not 4- or 5-dependent.

Now, in a sequence of examples, the data dependence measures as given by Definition 7 will be analyzed for certain computational problems. The results are collected in Table 6, i.e., the following examples may be considered as explanatory remarks to this table.

Example 8. The *multiplication of two* $N \times N$ *real matrices* may be considered as a $2N^2$-ary function into the set of N^2-tuples of real numbers. For this computational problem, it is evident that

$$\lambda_{\text{MATRIX-MULTIPLICATION}} = 2N,$$

$$\gamma_{\text{MATRIX-MULTIPLICATION}} = 2N^2, \quad \text{and} \quad \tau_{\text{MATRIX-MULTIPLICATION}} = 2N^3,$$

where these maximal values of data dependence are true for each input vector of length $2N^2$ containing non-zero values in all positions. By this example it follows that the upper bounds (7.2) and (7.3) cannot be reduced in general. The *inversion of an* $N \times N$ *real matrix in place* may be considered as an N^2-ary function into the set of N^2-tuples of real numbers. We have

$$\lambda_{\text{MATRIX-INVERSION-IP}} = \gamma_{\text{MATRIX-INVERSION-IP}} = N^2,$$

and

$$\tau_{\text{MATRIX-INVERSION-IP}} = N^4,$$

where this maximal case of data dependence appears for any matrix containing non-zero values in all N^2 positions. These data depence quantities may be considered as a direct consequence of the data dependence quantities for the determinant *of an* $N \times N$ *real matrix,*

$$\lambda_{\text{DETERMINANT}} = \gamma_{\text{DETERMINANT}} = \tau_{\text{DETERMINANT}} = N^2.$$

The solution of a *system of N linear equations* in N unknowns may be considered as an (N^2+N)-ary function into the set of N-tuples of real numbers. We obtain

$$\lambda_{\text{LINEAR—EQUATIONS}} = \gamma_{\text{LINEAR—EQUATIONS}} = N^2+N,$$

and

$$\tau_{\text{LINEAR—EQUATIONS}} = N^3+N^2.$$

Transposing an $N\times N$ real matrix in place may be considered as an N^2-ary function into the set of N^2-tuples of real numbers,

$$\lambda_{\text{TRANSPOSITION—IP}} = 1, \quad \text{and} \quad \gamma_{\text{TRANSPOSITION—IP}} = \tau_{\text{TRANSPOSITION—IP}} = N^2,$$

but for *binary operations on permutated $N\times N$ real matrices in place,*

$$(a_{ij})_{i,j=0,1,\dots,N-1} \Rightarrow (\mathrm{op}_2(a_{ij}, a_{\pi(i,j)}))_{i,j=0,1,\dots,N-1},$$

considered as N^2-ary functions into the set of N^2-tuples of real numbers,

$$\lambda_{\text{MATRIX—}\pi\text{—IP}} = 2 \quad \text{for} \quad \pi \neq id,$$

$$\gamma_{\text{MATRIX—}\pi\text{—IP}} = N^2,$$

and

$$\tau_{\text{MATRIX—}\pi\text{—IP}} = 2N^2 - \mathrm{card}\,\{(i,j): 0 \leqq i,j \leqq N-1 \;\&\; \pi(i,j) = (i,j)\},$$

the transposition may be considered as a special permutation π^*, $\tau_{\text{MATRIX—}\pi^*\text{—IP}} = 2N^2-N$, and op_2 as the exchange operation in this case, $\mathrm{op}_2(a_{ij}, a_{\pi^*(i,j)}) = (a_{\pi^*(i,j)}, a_{ij})$, where the second component of these resulting tuples will be considered as a dummy result.

Example 9. In this example, three two-dimensional transforms of $N\times N$ pictures will be dealt with. First, the *Fourier transform of an $N\times N$ complex matrix* (2D-DFT, two-dimensional discrete Fourier transform, cp. [9]) may be considered as a $2N^2$-ary function into the set of $2N^2$-tuples of real numbers. In this case, we have

$$2N^2-4 \leqq \lambda_{\text{2D—DFT}} \leqq 2N^2-1,$$

$$\gamma_{\text{2D—DFT}} = 2N^2, \quad \text{and} \quad 2N^4 \leqq \tau_{\text{2D—DFT}} \leqq 4N^4-2N^2,$$

where these maximal values of data dependence are true for each input vector of length $2N^2$ containing non-zero values in all positions. For the exact determination of $\lambda_{\text{2D—DFT}}$ and $\tau_{\text{2D—DFT}}$, the influence of different values of N has to be studied. The *Walsh transform of an $N\times N$ real matrix* (2D-WT, two dimensional Walsh transform, cp. [9]) may be considered as an N^2-ary function into the set of N^2-tuples of real numbers,

$$\lambda_{\text{2D—WT}} = \gamma_{\text{2D—WT}} = N^2, \quad \text{and} \quad \tau_{\text{2D—WT}} = N^4,$$

where these maximal values of data dependence are true for any input vector of length N^2. The computation of the *parallel Roberts gradient* (see Example 1) on images of size $M\times N$ may be considered as an MN-ary function into the set of MN-tuples of real numbers. For this function,

$$\lambda_{\text{ROBERTS—GRADIENT}} = 4,$$

$$\gamma_{\text{ROBERTS—GRADIENT}} = MN, \quad \text{and} \quad \tau_{\text{ROBERTS—GRADIENT}} = 4MN-2M-2N-2,$$

by considering the case of non-zero values in all MN positions, and by paying attention to border effects.

Example 10. The computation of the *convex hull of a simple polygon,* cp. [5]’ where the N extreme points of the polygon are given by coordinate tuples of real numbers starting with the uppermost-leftmost point, may be considered as a $2N$-ary function into the set of $2N$-tuples of real numbers. In the resulting vector of length $2N$, there appear all coordinate tuples of the extreme points of the convex hull of the given polygon in order, starting with the uppermost-leftmost point, and with the same run orientation as the given polygon. Positions actually not needed in this resulting $2N$-tuple contain value zero by assumption. In this case, it follows that

$$\lambda_{\text{CH—SIPOL}} = \gamma_{\text{CH—SIPOL}} = 2N, \quad \text{and} \quad 2N^2-8N+12 \leqq \tau_{\text{CH—SIPOL}} \leqq 4N^2$$

by analyzing the input situation of special convex polygons with N extreme points as illustrated in Fig. 2, for $N \geqq 4$. The computation of the *convex hull of N planar points,* cp. [5], given by coordinate tuples of real numbers, may be considered as a $2N$-ary function into the set of $2N$-tuples of real numbers as described above, analogously to the simple polygon situation. For this problem,

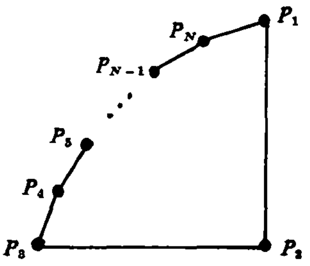

Figure 2.
Convex polygon for analyzing the maximal possible data dependence situation, for $N \geqq 4$

$$\lambda_{\text{CH—POINT}} = \gamma_{\text{CH—POINT}} = 2N, \quad \text{and} \quad \tau_{\text{CH—POINT}} = 4N^2,$$

where these maximal values are true for any input situation. The computation of the *Voronoi diagram of N planar points,* cp. [5], given by coordinate tuples of real numbers, may be considered as a $2N$-ary function into the set of $(18N-33)$-tuples of real numbers in the following sense. The Voronoi diagram may have $2N-5$ vertices at most, and, as a special planar graph, $3N-6$ edges at most, for $N \geqq 3$. See Fig. 3 for an illustration of the construction of such a “maximal Voronoi diagram”, where the number $v(N)$ of vertices, and the number $e(N)$ of edges satisfy the recursive equations

$$v(3) = 1, \quad e(3) = 3,$$

$$v(N+1) = v(N)+2, \quad \text{and} \quad e(N+1) = e(N)+3$$

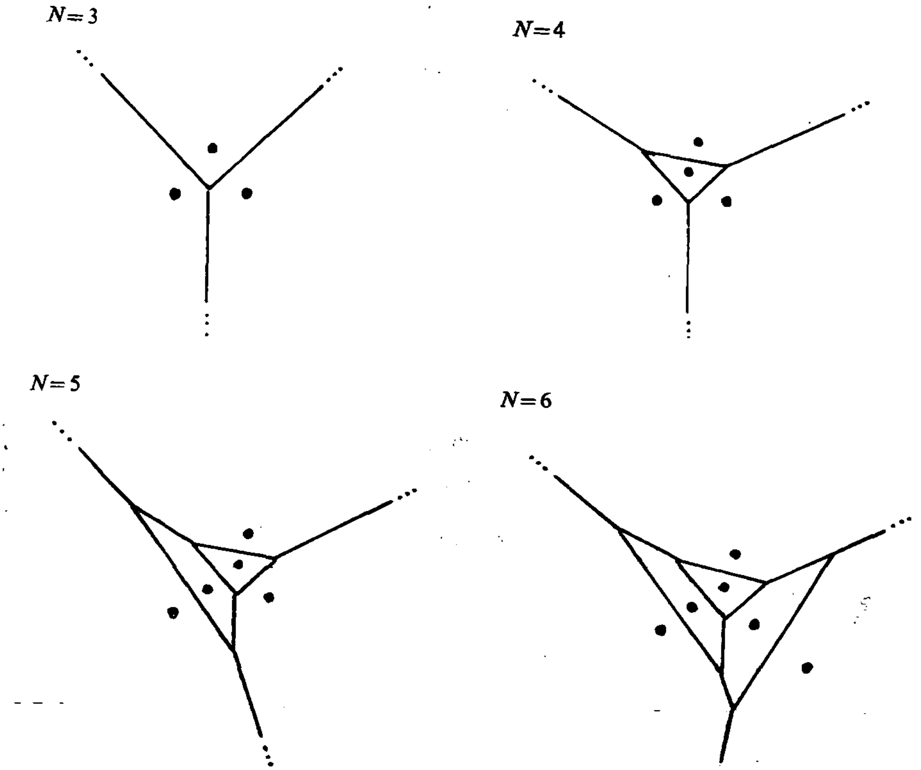

Figure 3.
Voronoi diagrams for $N=3, 4, 5, 6$ with $2N-5=1, 3, 5, 7$ vertices and $3N-6=3, 6, 9, 12]$ edges, respectively

for $N \geqq 3$. The $18N-33=3(2N-5)+4(3N-6)$ positions of the resulting vector of a Voronoi diagram computation we consider as a unique characterization of a Voronoi diagram by linearization of adjacency lists for this special graph structure with the positions for each vertex where two are reserved for the coordinate values and one for a common pointer, and two times two positions for each edge — for the index of the vertex at the other end of the edge, of for the slope of the edge, and for a common pointer. For concrete inputs of N points, positions actually not needed in the resulting $(18N-33)$-tuple contain value zero by assumption. Then, we have

$$\lambda_{\text{VORONOI--DIAGRAM}} = \gamma_{\text{VORONOI--DIAGRAM}} = 2N,$$

and

$$12N-3 \leqq \tau_{\text{VORONOI--DIAGRAM}} \leqq 2N(18N-33),$$

for $N \geqq 3$, where the local and global case may be analyzed by using a regular N-gon, and for the total case a Voronoi diagram in the sense of Fig. 3, with $2N-5$ points, was used where each point of the diagram essentially depends on there input points, i.e., on six coordinate values.

Example 11. *Matching of a pattern* of length M against a string of length $N(M \leqq N$ and the elements of pattern and string are assumed to be reals) may be considered as a $(N+M)$-ary function into the set of $(N-M+1)$-tuples on $\{0, 1\}$ where, for

$$f_{\text{PATTERN—MATCHING}}(p_1, p_2, \ldots, p_m;\ s_1, s_2, \ldots, s_m) = (e_1, e_2, \ldots, e_{N-M+1})$$

we have $e_i = 1$ iff $s_{i+j} = p_{j+1}$, for all $j = 0, 1, \ldots, M-1$, and $e_i = 0$ otherwise, for $i = 1, 2, \ldots, N-M+1$. We have

$$\lambda_{\text{PATTERN—MATCHING}} = 2M,$$

$$\gamma_{\text{PATTERN—MATCHING}} = M+N, \quad \text{and} \quad \tau_{\text{PATTERN—MATCHING}} = 2M(N-M+1).$$

In all three cases, the maximal dependence may be analyzed for the trivial input situation $p_i = s_j = \text{const}$, for $i = 1, 2, \ldots, M$ and $j = 1, 2, \ldots, N$. *Detection of a pattern* of length M within a string of length N, $M \leqq N$, may be considered as an $(N+M)$-ary function into the set $\{0, 1\}$ where the output is equal to $\max\{e_i : i = 1, 2, \ldots, N-M+1\ \&\ f_{\text{PATTERN—MATCHING}}(p_1, p_2, \ldots, p_M;\ s_1, s_2, \ldots, s_N) = (e_1, e_2, \ldots, e_{N-M+1})\}$ for input $(p_1, p_2, \ldots, p_M;\ s_1, s_2, \ldots, s_N)$. Then,

$$\max\{2M, M + \lfloor N/M \rfloor\} \leqq \lambda_{\text{PATTERN—SIGNALIZATION}} \leqq M+N.$$

Note that this represents the first example of a computational problem where the equality $\gamma_f = n$ remains an open problem, for an n-ary function f with $n = N+M$ in the case of pattern detection. As a last example, *sorting of N real numbers* may be considered as an N-ary function into the set of N-tuples of real numbers. For this very important problem, we have

$$\lambda_{\text{SORTING}} = \gamma_{\text{SORTING}} = N, \quad \text{and} \quad \tau_{\text{SORTING}} = N^2,$$

where these maximal values are true for N pairwise different input values.

4. Data transfer lemma and applications

Between the quantitative descriptions of data transfer for SIMD systems (Section 2) and of data dependence for computational problems (Section 3), the following direct relation holds.

Lemma 1. (Data Transfer Lemma). Let SYS∈SIMD, and let π be an arbitrary program for SYS for the computation of a function f which is n-ary and has m-tuple values. Let R denote the set of output registers of SYS where the m-tuples appear at the end of the computation $(\text{card}(R) = m$, off-line mode$)$, or those output registers of SYS via which the computed values of the m-tuples leave SYS in certain waves of information $(\text{card}(R) \leqq m$, on-line mode$)$. Then, the computation of $f(x_1, x_2, \ldots, x_n)$ on SYS by π requires at least t_0 steps of comdutation for a given input $(x_1, x_2, \ldots, x_0) \in \mathit{domain}(f)$, where $\Lambda_{\text{SYS}}(t_0) \geqq \lambda_f$, $\Gamma_{\text{SYS}}(\text{card}(R), t_0) \geqq \gamma_f$, and $T_{\text{SYS}}(\text{card}(R), t_0) \geqq \tau_f$.

Proof. Let us consider the local off-line or on-line situation. Assume that $\lambda_f = \text{card}(\text{sub}_i(x_1, x_2, \ldots, x_n))$, for a given input vector $(x_1, x_2, \ldots, x_n)$, and for a given position i, $1 \leqq i \leqq m$. Let $\mathrm{sub}_i(x_1, x_2, \ldots, x_n) = \{j_1, j_2, \ldots, j_{\lambda_f}\}$. For any position i_k, $k = 1, 2, \ldots, \lambda_f$, either the name of an input register receiving value x_{j_k} at a given moment will be transfered to the receptive field $\mathrm{rec}_\pi^{(x_1, x_2, \ldots x_n)}(r^{(i)}, t^*)$ by some operational instructions only, if value $\mathrm{proj}_i(f(x_1, x_2, \ldots, x_n))$ appears in register $r^{(i)} \in R$ at time $t^* \leqq t_0$ of computation, or during the t^* steps of computation of $\mathrm{proj}_i(f(x_1, x_2, \ldots, x_n))$ at least one test instruction JGTZ, JZERO, or JLTZ must be performed where the contents of the CPU accumulator depends on the input value x_{j_k} at the moment of testing. In the second case, if the test instruction is followed by certain operational instructions directed to register $r^{(i)}$ the name of the input register receiving value x_{j_k} at a given moment will be transferred to the receptive field $\mathrm{rec}_\pi^{(x_1, x_2, \ldots, x_n)}(r^{(i)}, t^*)$, too; cp. (iv) in Definition 4. Without loss of generality, assume that $j_1, j_2, \ldots, j_v$, $v \leqq \lambda_f$, denote all the positions which have produced register names in the receptive field $\mathrm{rec}_\pi^{(x_1, x_2, \ldots, x_n)}(r^{(i)}, t^*)$. If $v = \pi_f$, then $\pi_f \leqq \mathrm{card}\left(\mathrm{rec}_\pi^{(x_1, x_2, \ldots, x_n)}(r^{(i)}, t^*)\right) \leqq \lambda_{\mathrm{SYS}}(t_0)$ follows immediately. For $v < \lambda_f$, let $t_1, t_2, \ldots, t_w$ be all the moments where test instructions have to be performed according to π and input $(x_1, x_2, \ldots, x_n)$ such that the contents of the CPU accumulator depend on one of the input values $x_{j_{v+1}}, \ldots, x_{j_{\lambda_f}}$ at least, at the moments of testing. Consider the following program π' computing something unspecified, produced by π and $(x_1, x_2, \ldots, x_n)$ in the following way:

- all test instructions at moments $t_1, t_2, \ldots, t_w$ will be deleted in π, and
- all other instructions of π will be performed according to π and input $(x_1, x_2, \ldots, x_n)$, in the same order, where all instructions LOAD α or $\mathrm{OP}_1\ \alpha$, for α equal to $=x, m, {}^*m$, or (j), will be replaced by $\mathrm{OP}_2\ \alpha$, for the same value of α, if such instructions appear in π.

Thus, the receptive field of register 0, i.e., the CPU accumulator, will increase monotonically according to π' and $(x_1, x_2, \ldots, x_n)$. After $t^* - w$ operations according to π', $\mathrm{rec}(0, t^* - w)$ contains all input register names for the input data $x_{j_{v+1}}, \ldots, x_{j_{\lambda_f}}$. This receptive field will be combined with $\mathrm{rec}_\pi^{(x_1, x_2, \ldots, x_n)}(r^{(i)}, t^* - w) \geqq \mathrm{rec}_\pi^{(x_1, x_2, \ldots, x_n)}(r^{(i)}, t^*)$ at moment $t^* - w + 1 \leqq t^*$ by adding an instruction $\mathrm{OP}_2\ \alpha$ (see conditions (OFF.2) and (ON.6)) or $\mathrm{OP}_2(j)$ (see conditions (OFF.4) and (ON.7)) to π'. Thus, $\lambda_f \leqq \mathrm{card}\left(\mathrm{rec}_\pi^{(x_1, x_2, \ldots, x_n)}(0, t^* - w + 1)\right) \leqq \leqq \Lambda_{\mathrm{SYS}}(t^* - w + 1) \leqq \Lambda_{\mathrm{SYS}}(t_0)$. Note that the off-line or on-line I/O convention is necessary to ensure that a non-accumulator PE register $r^{(i)}$ may be replaced by the accumulator of the same PE which is an output register, too. For this replacement, parallel STORE instructions may be replaced by parallel OP_1 instructions using the same masks for PE addresses.

What we have explained is one of the possible ways to ensure the necessary data transfer within time limit t_0, for the local off-line or on-line situation. The essential point in the program transformation from π to π' may be characterized by the word "linearization", because all test instructions could be deleted, in fact. This linearization approach may be used for the local, global and total situation in the following way.

For the given program π and an input situation I, all the performed instructions will be written as a linear sequence S_0. We obtain sequence S_1 by deletion of all instructions JLTZ, JZERO, JGTZ, JUMP, WRITE, and HALT in sequence S_0. Now, for the special case of an on-line program, if in sequence S_0 there were some STORE instructions in front of a WRITE instruction directed to certain output registers $r \in R$, then these STORE instructions will be shifted to the end of sequence S_1. In the resulting sequence S_2, all serial or parallel $\mathrm{OP}_1\,\alpha$ or LOAD α instructions will be replaced by an $\mathrm{OP}_2\,\alpha$ instruction formally, in the same position for the same value of α. For the resulting sequence S_3 we have monotonically increasing receptive fields for all accumulators, for the CPU and PEs. Also, by the described step from S_1 to S_2, for sequence S_3 the receptive fields of output registers will be monotonically increasing for consecutive output waves of information. Now, if in the original sequence S_0 there was no test instruction, our program linearization is finished. In the other case, in S_3 we shall place an instruction JZERO, e.g., in that position where the last test instruction was located in sequence S_0. Now consider an arbitrary output register $r \in R$. If there is an operational instruction behind the JZERO instruction directed to r then register r will obtain the receptive field of the CPU accumulator containing all the register names corresponding to tested input values, cp. (iv) in Definition 4. If there is no operational instruction behind the JZERO instruction directed to r then we shift the last instruction directed to r in front of the JZERO instruction to a position behind this instruction. By consideration of all registers $r \in R$, our program linearization is finished. Note that the length of the resulting linear instruction sequence is restricted by the length of the original sequence S_0.

Now assume that $\lambda_f = \mathrm{card}\,(\mathrm{sub}_i(x_1, x_2, \ldots, x_n))$ for a certain i, $1 \leqq i \leqq n$, $\gamma_f = \mathrm{card}\,(\bigcup_{i=1}^{m} \mathrm{sub}_i(y_1, y_2, \ldots, y_n))$ and $\tau_f = \sum_{i=1}^{m} \mathrm{card}\,(\mathrm{sub}_i(z_1, z_2, \ldots, z_n)$, for certain input vectors $(x_1, x_2, \ldots, x_n)$, $(y_1, y_2, \ldots, y_n)$, $(z_1, z_2, \ldots, z_n)$. These input vectors characterize input situations I_x, I_y, I_z for SYS. By linearization of π according to these input situations we obtain linear programs π_x, π_y, π_z, respectively, all of length $\leqq t_0$. Thus, we have

$$\lambda_{\pi_x}^{(x_1, x_2, \ldots, x_n^T)}(R, t_0) \geqq \lambda_f,$$

$$\gamma_{\pi_y}^{(y_1, y_2, \ldots, y_n)}(R, t_0) \geqq \gamma_f,$$

$$\tau_{\pi_z}^{(z_1, z_2, \ldots, z_n)}(R, t_0) \geqq \tau_f,$$

which proves our statements. □

Corollary 1. Let $\mathrm{CLASS} \subseteqq \mathrm{SIMD}$. For any system $\mathrm{SYS} \in \mathrm{CLASS}$, the computation of a function f which is into the set of m-tuples of real numbers requires at least t_0 steps of computation in the worst case, where $\Lambda_{\mathrm{CLASS}}(t_0) \geqq \lambda_f$, $\Gamma_{\mathrm{CLASS}}(m, t_0) \geqq \gamma_f$, and $T_{\mathrm{CLASS}}(m, t_0) \geqq \tau_f$.

Proof. Immediately by Lemma 1 where the generalization about all programs computing the function f is used as well as about all systems of CLASS. For the on-line case note that there may already be a certain $m_0 \leqq m$ such that $\Gamma_{\mathrm{CLASS}}(m_0, t_0) \geqq \gamma_f$, and $T_{\mathrm{CLASS}}(m_0, t_0) \geqq \tau_f$. □

Example 12. Let CLASS = {EXAMP1} and consider the computation of the parallel Roberts gradient as described in Example 1. In this case we get the trivial lower time bound 1 only; an upper bound was 29. Now, let CLASS = {EXAMP3} and consider the computation of the arithmetical averages of M consecutive waves of information of length $N = 2^{n-1}$ as described in Example 3. Here, by Corollary 1 we obtain the lower time bound $n+2M-2=\max\{n-1,\ n+2M-2,\ n+M-1\}$, cp. equation (6.1), (6.2), (6.3), for values $\lambda_f=N, \gamma_f=N\cdot M$ and $\tau_f=N\cdot M$. An upper bound was $6M+n$.

Using common asymptotic notations, for both examples the optimal times $\theta(1)$ and $\theta(M+n)$ are known as a result.

Theorem 3. For any system SYS$\in$OFF-NET$_p$, $p\geqq 2$, the computation of a function f which is into the set of m-tuples of real numbers requires at least t_0 steps of computation in the worst case, where

$$t_0 \geqq \max\{(d_1-1)/2,\ (d_2-m)/2m,\ (d_3-m)/2m\}$$

for $p=2$, and for $p\geqq 3$

$$\begin{aligned} t_0 \geqq \max\big\{&\log_{p-1}(d_1(p-2)+2)-1.586,\\ &\log_{p-1}(d_2(p-2)+2)-\log_{p-1}m-1.586,\\ &\log_{p-1}(d_3(p-2)+2)-\log_{p-1}m-1.586\big\}, \end{aligned}$$

if f is locally d_1-dependent, globally d_2-dependent, and totally d_3-dependent.

Proof. Immediately by Theorem 1, Definition 7 and Corollary 1 where the relation $\log_{p-1}p > 1.586$, $p\geqq 3$, was used. □

In Table 7 are collected, for the classes of off-line systems defined in Section 1, the lower time bounds that may be obtained by using Corollary 1. Because the classes OFF-LINEAR, OFF-PS, OFF-BINTREE and OFF-QUADTREE represent examples for the maximal transfer situation as characterized by Theorem 1, for these classes the lower time bounds are as given by Theorem 3. If a function f into the set of m-tuples is globally or totally d'-dependent, then the value d has to be replaced by d'/m in the lower time bounds given in Table 7, to obtain the corresponding values for the global or total situation.

Theorem 4. For any system SYS$\in$ON-NET$_{p,q}$, $2\leqq p<\infty$, $1\leqq q<p$, the computation of a function f which is into the set of m-tuples of real numbers requires at least t_0 steps of computation in the worst case, where

$$t_0 \geqq \max\{(d_1+1)/2,\ (d_2+m)/2m,\ (d_3+m)/2m\}$$

for $q=1$, and for $q\geqq 2$

$$\begin{aligned} t_0 \geqq \max\big\{&\log_q(d_1(q-1)+1),\quad \log_q(d_2(q-1)/m+1),\\ &\log_q(d_3(q-1)/m+1^{T}\big\}, \end{aligned}$$

if f is locally d_1-dependent, globally d_2-dependent, and totally d_3-dependent.

Proof. Immediately by Theorem 2, Definition 7 and Corollary 1. □

In Table 8 are collected, for the classes of on-line systems defined in Section 1, the lower time bounds that may be obtained by using Corollary 1. Because the classes ON-LINEAR$_{\{0\}}$, ON-BINTREE$_{\{1,2\}}$, and ON-QUADTREE$_{\{1,2,3,4\}}$ represent examples for maximal transfer situations as characterized by Theorem 2, for these classes the lower time bounds are as stated by Theorem 4. As in the case of Table 7, if a function f into the set of m-tuples is globally or totally d'-dependent, then the value d has to be replaced by d'/m in the lower time bounds given in Table 8, for obtaining the corresponding values for the global or total situation. Note that value m may be replaced by a value $m_0 \leqq m$ for special ON-NET systems.

5. Conclusions

In this paper we have given a general framework for the description of parallel processing systems, and explained how data flow may be used for analyzing lower time bounds in general. Note that this approach may be applied to supercomputers as well as to on-chip realizations. Problems connected with the technical features

Table 6. Local, global and total data dependence measures

Computational problem f	n	m	λ_f	γ_f	τ_f
MATRIX MULTIPLICATION	$2N^2$	N^2	$2N$	$2N^2$	$2N^3$
MATRIX INVERSION IP	N^2	N^2	N^2	N^2	N^4
DETERMINANT	N^2	1		N^2	
LINEAR EQUATIONS	N^2+N	N	N^2+N	N^2+N	N^3+N^2
TRANSPOSITION IP	N^2	N^2	1	N^2	N^2
MATRIX π IP	N^2	N^2	2 for $\pi \neq id$	N^2	$2N^2 - \# \{(i, j): \pi(i, j) = (i, j)\}$
2D—DFT	$2N^2$	$2N^2$	$\geqq 2N^2-4$ $\leqq 2N^2-1$	$2N^2$	$\geqq 2N^4$ $\leqq 4N^4-2N^2$
2D—WT	N^2	N^2	N^2	N^2	N^4
ROBERTS GRADIENT	MN	NM	4	MN	$4MN-2M-2N-2$
CH SIPOL	$2N$	$2N$	$2N$	$2N$	$\geqq 2N^2-8N+12$ $\leqq 4N^2$
VORONOI DIAGRAM	$2N$	$18N-33$	$2N$	$2N$	$\geqq 12N-30$ $\leqq 36N^2-66N$
PATTERN MATCHING	$N+M$	$N-M+1$	$2N$	$M+N$	$2M(N-M+1)$
PATTERN SIGNALIZATION	$N+M$	1	$\geqq \max \{2M, M+\lfloor N/M \rfloor\}$,		$\leqq M+N$
SORTING	N	N	N	N	N^2

of architecture elements were by passed by the selected level of abstract system description. Thus, in the discussion of parallel algorithms for a given model SYS∈ ∈SIMD we may have in mind quite different technical implementations, but we may discuss parallel algorithms for all of them at once using the abstract model SYS∈SIMD. For example, an important problem is given by the necessary decision between different structures of parallel processing systems to ensure efficient algorithmic solutions for classes of computational problems such as mentioned in Example 8 (matrix-type computations), 9 (two-dimensional transforms), 10 (geometric problems), or 11 (combinatorial problems). According to our considerations in [4] the selection of parallel algorithms crucially depends on the given parallel processing system and comparisons between different SIMD systems on the basis of knowledge about optimal algorithms represents quite a hard task. Also, there are nearly as many different models for parallel processing as papers on this topic, making comparative studies of different parallel structures nearly impossible. In the present paper an attempt was made to propose a classification of special parallel processing systems which have been of widespread interest in the past. The proof of the practicability of the proposed exact definition of SIMD systems will be the subject of forthcoming papers; the first programs of the PARSIS project fit well into this framework.

By using Tables 6, 7, and 8 the interested reader may obtain lower time bounds for different combinations of SIMD systems and computational problems, e.g., the lower time bound $\log_2(N^2+1)$ for the two-dimensional Walsh transform on

Table 7. Lower time bounds for off-line systems in OFF-CLASS for computing a local d-dependent function

CLASS	p	lower time bound	$d=128$	$d=128^2$
LINEAR	2	$(d-1)/2$	64	8, 192
HEXAGONAL	3	$\left(\left(\frac{8}{3}d-\frac{5}{3}\right)^{1/2}-1\right)\Big/2$	9	105
SQUARE or ILLIAC	4	$((2d-1)^{1/2}-1)/2$	8	91
TRIAGONAL	6	$\left(\left(\frac{4}{3}d-\frac{1}{3}\right)^{1/2}-1\right)\Big/2$	7	74
DIAGONAL	8	$(d^{1/2}-1)/2$	6	64
PS	3	$\log_2(d+2)-1.586$	6	13
BINTRE top node	3	$\log_2(d+2)-1.586$ $\log_2(d+1)-1$	6 7	13 14
TRIANGLE top node	5	$t_0 \geqq \log_2(d-t_0^2+2t_0+5)-2.586$ $\log_2(d+1)-1$	5 7	12 14
QUADTREE top node	5	$\log_4(3d+2)-1.161$ $\log_4(3d+1)-1$	4 5	7 7

Table 8. Lower time bounds for on-line systems in ON-CLASS for computing a local d-dependent function

CLASS	p	$\{i_1, \ldots, i_q\}$	Lower time bound	$d=128$	$d=128^2$
LINEAR	2	$\{0\}$	$(d+1)/2$	65	8,193
HEXAGONAL	3	$\{0, 1\}$	$((8d+1)^{1/2}-1)/2$	16	181
SQUARE or ILLIAC	4	$\{0, 1, 2\}$	$d^{1/2}$	12	128
TRIAGONAL	6	$\{0, 1, 2, 3, 4\}$	$\left(\left(\frac{8}{5}d-\frac{3}{5}\right)^{1/2}-1\right)\Big/2$	7	81
DIAGONAL	8	$\{0, 1, 2, 3, 4, 6, 7\}$	$\left(\left(\frac{8}{7}d-\frac{3}{7}\right)^{1/2}-1\right)\Big/2$	6	64
BINTREE	3	$\{1, 2\}$	$\log_2(d+1)$	8	15
TRIANGLE	5	$\{1, 2, 3, 4\}$	$\log_2(d+1)$	8	15
QUADTREE	5	$\{1, 2, 3, 4\}$	$\log_4(3d+1)$	5	8
PS	3	$\{0, 1\}$	$f_{t_0+2} \geq d+2$ for the Fibonacci numbers $f_0, f_1, f_2, \ldots$	11	21

ON-TRIANGLE systems. The characterization of data dependencies for computational problems as given by Definition 7 may be refined, e.g., by consideration of changes of function values not only by changing arguments in one position but in several positions.

Abstract

Starting with an exact definition of classes of SIMD (single instruction, multiple data) systems, a general approach to obtaining lower time bounds by data flow analysis is presented. Several interconnection schemes, such as the square net, the perfect shuffle, the infinite binary tree, etc. are analyzed with respect to their data transfer possibilities. For some types of computational problems the data dependencies are analyzed in a quantitative way. From both types of analysis, lower time bounds result for many combinations of SIMD systems and computational problems, for example, $O(\log N)$ for on-line quadtree-net systems and the computation of Voronoi diagrams for N planar points, $O(N)$ for off-line diagonal-net systems and the two-dimensional discrete Fourier transform, and $O(\sqrt{N})$ for off- or on-line Illiac-net systems and sorting of N items.

CENTER FOR AUTOMATION RESEARCH
UNIVERSITY OF MARYLAND
COLLEGE PARK, MD 20742
U.S.A.

* PERMANENT ADDRESS:
FRIEDRICH SCHILLER UNIVERSITY
DEPARTMENT OF MATHEMATICS,
UNIVERSITY TOWER 17TH FLOOR,
DDR-6900 JENA,
GERMAN DEMOCRATIC REPUBLIC

The support of the U. S. Air Force Office of Scientific Research under Grant AFOSR-77-3271 is gratefully acknowledged, as is the help of Janet Salzman in preparing this paper. The author thanks the government of the German Democratic Republic for financial support and Azriel Rosenfeld for his efforts in making the author's stay in College Park possible and effective as well.

References

[1] H. Abelson, Lower bounds on information transfer in distributed computations, J. ACM *27* (1980), 384—392.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA (1974).

[3] W. M. Gentleman, Some complexity results for matrix computation on parallel processors, J. ACM *25* (1978), 112—115.

[4] R. Klette, Zeitkompliziertheit von Berechnungsproblemen der digitalen Bildverarbeitung — Vergleiche zwischen sequentieller und paralleler Datenverarbeitung (in Slovakian, to appear, VEDA Publish. House, Bratislava).

[5] R. Klette, Geometrische Probleme der digitalen Bildverarbeitung, BILD UND TON *35* (1982), 101—110.

[6] R. Klette and R. Lindner, Zweidimensionale Vektormaschinen und ihr Leistungsvermögen bei der Lösung von Entscheidungsproblemen der Aussagenlogik, EIK *15* (1979), 37—46.

[7] T. Legendi, A cellular processor project, International Workshop on Parallel Processing by Cellular Automata, Berlin, GDR, Sept. 15—16, 1982.

[8] V. R. Pratt and L. J. Stockmeyer, A characterization of the power of vector machines, J. Computer System Sciences *12* (1976), 118—121.

[9] A. Rosenfeld and A. C. Kak, *Digital Picture Processing* (Second Ed.), Academic Press, New York (1982).

[10] H. J. Siegel, A model of SIMD machines and a comparison of various interconnection networks, IEEE Trans. Computers *C-28*, (1979), 907—917.

Received May 13, 1983.