A METHOD FOR TRANSLATING PRAM-OPTIMAL ALGORITHMS TO AT^2 INTERCONNECT-OPTIMAL CIRCUITS

A PREPRINT

April 10, 2020

ABSTRACT

We give a method that under some conditions translates PRAM-optimal models into AT^2 Interconnect-optimal hardware circuits. The key insight here is that many of the most applicable problems have AT^2 Interconnect complexity of $A = \Omega(N), T = \Omega(\sqrt{N})$ (where N is some measure of problem size); further, the same problems are parallelizable, i.e. there exist PRAM algos that solve them and use $\leq N$ processors and require $O(\log^k N)$ time. Thus, Theorem 2.5 gives a methodical way to translate PRAM-optimal designs into a systolic array which will have optimal area and optimal time (up to logarithmic factors). Notably, under some additional requirements, we are able to map the design without any multiplicative logarithmic overhead.

This result stands in contrast to the criticism of the PRAM model as a model with unrealistic assumptions. In fact, we show that for a large class of functions PRAM-optimal models are actually optimal under the AT^2 Interconnect metric - a metric specifically created to realistically model on-chip designs.

Finally we show similar optimal translations can be obtained under the even more restrictive Dissipative AT^2 Interconnect model.

***K*eywords** First keyword · Second keyword · More

1 Preliminaries

1.1 AT^2 Interconnect Model

We use the computational model defined in [1]. Here it is:

Definition 1.1. *A **computational device** is a circuit that computes a Boolean function $(y_1, y_2, ...) = F(x_1, x_2, ...)$. Information is treated digitally and is communicated to the device through I/O ports in a fixed format. With each variable x_i (respectively, y_i) there is associated both an input (respectively, output) port and a chronological rank on the set of inputs (respectively, outputs). (We say the computation is **when-** and **where-determinate**.) A computational device is internally a circuit connecting nodes and wires into a directed graph and is defined by the planar geometrical layout of this graph with at most a constant number of crossovers between edges. The minimum width of a wire carrying a bit of data is some constant λ and all nodes occupy some constant area. We assume inputs can be time multiplexed. We also assume that the times and locations of the I/O bits are fixed and independent of input values.*

Two parameters characterize the computational complexity of a circuit: its *area* A and its *time of computation* T.

Definition 1.2. *For a fixed value of the inputs, T measures the **time** between the appearances of the first input bit and the last output bit. The maximum value of T for all possible inputs defines the time complexity of the circuit.*

Another important parameter is the *period* of a circuit.

Definition 1.3. *Since it is possible to pipeline the computations on several sets of inputs, we define the **period** P as the minimal time interval separating two input sets. More precisely, if $(a_1, ..., a_N)$ and $(b_1, ..., b_N)$ are two sets of inputs, and if the time separating the appearance of a_i and b_i is the same for all i, this interval defines the period P.*

Viewing the circuit as a directed graph, we associate with each node (including I/O ports) a Boolean variable $I_i(t)$ (respectively, $O_j(t)$) for each incoming (respectively, outgoing) wire, which is defined at all times t. These variables represent the information available at the entrance and exit points of a node.In addition, there corresponds to each node a state $S(t)$ chosen from among a finite number of possible states. Then each node of the circuit computes a function F of the form

$$[S(t+\tau), ..., O_j(t+\tau), ...] = F[S(t), ..., I_i(t), ...]. \tag{1}$$

We also assume that the time of propagation across a wire is at least proportional to the length of the wire. Let $I(t)$ and $O(t)$ be the variables associated with the ends of a wire of length L. We require that

$$I(t+T) = O(t) \quad \text{for} \quad T = \Omega(L). \tag{2}$$

Definition 1.4. *We say that there exists a* ***datapath*** *from an input variable x to a node V if there is a directed path to V from the input port where x is read in and that information be propagated from x to V for at least one input value.*

1.2 Relevant Asymptotic Results

Lemma 1.1. *If P is an arbitrary convex polygon with a perimeter N, the maximum distance between any vertex of P and an arbitrary point in the plane is $\Omega(N)$.*

Theorem 1.1. *It takes time $T = \Omega(\sqrt{N})$ to perform a fan-out of degree N. A fan-out of degree N refers to the distribution of N copies of an information bit at N different locations on the circuit.*

Definition 1.5. *A Boolean function $y = f(x_1, ..., x_N)$ is a fan-in of degree N if there is an assignment of the N variables such that for any i,*

$$f(a_1, ..., a_i, ..., a_N) \neq f(a_i, ..., \neg a_i, ..., a_N). \tag{3}$$

Any such N-tuple $(a_1, ..., a_N)$ of bits is called a ***hard input*** *of the function f.*

Definition 1.6. *A* ***fan-in*** *of N bits is a circuit that implements a boolean function on N hard inputs.*

Theorem 1.2. *If T (respectively, A) denotes the minimum time (respectively, area) for performing a fan-in of N inputs, we have $T = \Omega(\sqrt{N})$ and $AT = \Omega(N)$.*

Theorem 1.3. *If T is the time, P the period, and A the area required by any circuit to add two N-bit integers, then $T = \Omega(\sqrt{N})$, $AT = \Omega(N)$, $AT^2 = \Omega(N^2)$.*

1.2.1 Transitive Functions

Definition 1.7. *Let $f(x_1, ..., x_n, s_1, ..., s_m) = (y_1, ..., y_n)$ be a boolean function ($B^n \times B^m \rightarrow B^n$). Now, f* ***computes a permutation group*** *G* ***on*** *n* ***elements*** *if for all $g \in G$ there is an assignment $(\alpha_1, ..., \alpha_m)$ to the control bits $(s_1, ..., s_m)$ such that $f(x_1, ..., x_n, \alpha_1, ..., \alpha_m) = (x_{g(1)}, ..., x_{g(n)})$ for all $(x_1, ..., x_n)$. The inputs $(x_1, ..., x_n)$ are called the permutation inputs. The function f is called* ***transitive of degree*** *n if G is a transitive group, i.e., if for all $i, j \in \{1, ..., n\}$ there is a $g \in G$ such that $g(i) = j$.*

Examples of transitive functions include:

- Cyclic shift of N bits
- Product of two p-bit integers, $N = 2p$
- Convolution of two p-element vectors, $N = (2p+1)(2k + \log_2 p)$.
- Linear transform of a p-element vector, $N = p(2k + \log_2 p)$ (where transform coefficients are counted as inputs)
- Product of three $p \times p$ matrices, $N = p^2(2k + \log_2 p)$

When necessary, we assume that all numbers are presented on k bits, with $k \geq \log_2 p$.

Theorem 1.4. *Let $f : B^{n+m} \rightarrow B^n$ be transitive of degree n. Then f is $n/6$-inseparable, and thus $C_{det}(f) = \Omega(n^2)$.*

Theorem 1.5. *Computing a transitive function of degree N takes time $T = \Omega(\sqrt{N})$ and requires an area $A = \Omega(N)$ (under the AT^2 Interconnect metric).*

Definition 1.8. *Consider a line L partitioning the circuit into two parts C_1 and C_2, each producing half the output bits (more specifically, let C_1 and C_2 be balanced partitions). We define the* ***minimal cross-flow*** *I to be the minimal number of the maximum over all inputs of the number of bits crossing L, for all balanced partitions C_1 and C_2. (In notation of the AT^2 metric, we would say the circuit is I-inseparable)*

Lemma 1.2. *Any circuit computing a function with minimum cross-flow I requires time $\Omega(\sqrt{I})$.*

1.2.2 Communication Complexity and Lower Bound on Transitive Functions

We know present an AT^2 lower bound for a wide set of functions. This was work originally presented in [18].

Definition 1.9. *A **deterministic algorithm** is given by two response functions $r_L : X \times B* \to B$ and $r_R : Y \times B* \to B$ and two partial output functions $out_L : X \times B* \to A$, $out_R : Y \times B* \to C$, where $B = \{0, 1\}$ and $B* = \{\{b_i\}_{i \in \mathbb{N}} : b_i \in B$ for all $i\}$. A computation on input x, y is a sequence $w = w_1 w_2 ... w_k$ of bits such that:*

1. *$w_{2i+1} = r_L(x, w_1, ..., w_{2i})$ for all $i \geq 0$.*
2. *$w_{2i+2} = r_R(y, w_1, ..., w_{2i+1})$ for all $i \geq 0$.*
3. *$(x, w_1, ..., w_k) \in dom(out_L)$ and $(y, w_1, ..., w_k) \in dom(out_R)$ and this is the smallest such k.*

*k is called the **length of the computation** and is also denoted $k(x, y)$. A deterministic algorithm **computes** a function $f : X \times Y \to A \times C$ if for all $x \in X$, $y \in Y$ we have $f(x, y) = (out_L(x, w), out_R(y, w))$ with w a function of x, y and the deterministic algorithm as described above. This definition is due to [36].*

Definition 1.10. *The **complexity of an algorithm** Alg is defined as*

$$C(Alg, n) = \max\{k(x, y) : x \in X, y \in Y, |x| + |y| = n\}. \tag{4}$$

Definition 1.11. *The **complexity of a function** $f : X \times Y \to A \times C$ is defined by*

$$C_{def}(f, n) = \min\{C(Alg) : Alg \text{ computes } f\} \tag{5}$$

The ideas in communication complexity can be translated to the VLSI world as follows: A chip can be cut in half, with part of the input and output ports on one side, and part on the other. Then a certain amount of communication has to happen between the two parts of the chip before proper output values can be determined. This is now a communication complexity problem, and the lower bounds of communication complexity can further be used to achieve lower bounds in the AT^2 metric. One issue that might be addressed is that the cut of the chip may partition input and output ports in many different ways, whereas communication complexity assumes that the input and output partitions have been defined upfront in the definition of the target function f. We will address this below.

Definition 1.12. *Let $f : B^n \to B^m$ and $w \geq 0$. Function f is w-**inseparable** if for all **balanced partitions** X, Y of $[1, ..., n]$, i.e. $n/3 \leq |X|, |Y| \leq 2n/3$, and all partitions A, C of $[1, ..., m]$ we have $C_{def}(f_{X,Y,A,C}) \geq w$, where $f_{X,Y,A,C} : B^{|X|} \times B^{|Y|} \to B^{|A|} \times B^{|C|}$ is defined by partitioning the input and output bits into left and right input and output bits as given by partitions X, Y and A, C.*

Theorem 1.6. *If $f : B^n \to B^m$ is w-inseparable then $AT^2 = \Omega(w^2)$ for every chip computing f.*

Now we describe the crossing sequence method.

Definition 1.13. *Let $f : X \times Y \to B^m$ and let I, J be a partition of the output bits of f. f has w-**flow** if there is partial input $y \in Y$ such that f restricted to $X \times \{y\}$ in its domain and J in its range has more than 2^{w-1} different points.*

Theorem 1.7. *If f has w-flow then $C_{det}(f) \geq w$ and $AT^2 = \Omega(w^2)$.*

1.2.3 AT^2 Lower Bounds on Sorting

Theorem 1.8. *Any circuit sorting N k-bit numbers, with $k \geq 2\log_2 N$, has an area $A = \Omega(N)$ and takes time $T = \Omega(\sqrt{Nk})$. Moreover, this bound is achieved by the bitonic mesh sorting algorithm.*

1.2.4 AT^2 Lower Bounds on Matrix Multiplication

atrix Multiplication has good Operational Intensity ($\Omega(\sqrt{M})$ compared to $\Omega(\log M)$ for sorting). It seems that algorithms that achieve peak operational intensity for matrix multiplication have already been implemented, and tight bounds are known (including constant factors). [31][32]

Theorem 1.9. *Under the AT^2 Interconnect model, matrix multiplication is $A = \Omega(N)$, $T = \Omega(\sqrt{N})$.*

1.3 The Dissipative Model

This model puts additional assumptions on the AT^2 Interconnect model [19]. Specifically, the additional assumptions are:

- To switch a gate requires one unit of energy.
- Memorizing a bit requires a unit of energy per unit of time.
- The energy is supplied to the circuit through its planar boundary, and the density of energy at any point is bounded by a constant.

Lemma 1.3. *If N gates in a circuit are switched at the same time, their convex hull has a perimeter $\Omega(N)$. (versus $\sqrt{N}$ in the AT^2 interconnect case)*

Proof. Since all the power comes from outside the circuit and is transmitted on the plane, the power inside any convex region of the circuit is at most proportional to its perimeter. Since switching a gate requires a unit of energy, the result is straightforward. □

Theorem 1.10. *Any circuit of perimeter Π that computes a transitive function of degree N in time T satisfies $\Pi = \Omega(N)$ and $T = \Omega(N)$ (versus $T = \Omega(\sqrt{N})$ in the AT^2 interconnect case).*

Theorem 1.11. *In the dissipative model, the time T required to add two N-bit integers satisfies $T = \Omega(N^{2/3})$.*

1.4 The PRAM Model

Here are the basic assumptions of the PRAM model (Figure 1b) as described in [38]:

- The PRAM model assumes a collection of numbered processors that can access shared global memory and may also have some amount of local memory each.
- Each processor can access any shared memory cell in unit time.
- Inputs are stored in shared memory and outputs need to be stored back into shared memory (cold registers).

Since each processor has read and write access to each shared memory cell, the behavior of the model when there are read or write conflicts between multiple processors on a single memory cell give rise to three variations of the PRAM model:

- *Exclusive Read, Exclusive Write (EREW) PRAM*: no two processors are allowed to read or write the same shared memory cell simultaneously.
- *Concurrent Read, Exclusive Write (CREW) PRAM*: simultaneous read allowed, but only one processor can write.
- *Concurrent Read, Concurrent Write (CRCW) PRAM*. This model allows for concurrent writes with three variations:
  - Priority CRCW: processors assigned fixed distinct priorities, highest priority wins.
  - Arbitrary CRCW: one randomly chosen write wins.
  - Common CRCW: all processors are allowed to complete the write if and only if all the values to be written are equal.

1.5 Summary of Asymptotic Results from Previous Work

Algorithm	Time	Work	PRAM Model	AT^2 Interconnect Lowerbound
Prefix Sums	$O(\log N)$	$O(N)$	EREW	
Pointer Jumping	$O(\log h)$	$O(N \log h)$	CREW	
Merging	$O(\log N)$	$O(N)$	CREW	
Maximum	$O(\log \log N)$	$O(N)$	Common CRCW	$AT = \Omega(N), T = \Omega(\sqrt{N})$ (due to fan-in)
Fast Coloring	$O(\log N)$	$O(N \log N)$	EREW	
Parallel Search	$O\left(\frac{\log(N+1)}{\log(p+1)}\right)$	$O\left(\frac{N \log(N+1)}{\log(p+1)}\right)$	CREW	
Mutual Ranking	$O\left(\frac{(N+M) \log \log M}{p} + \log \log M\right)$	$O\left((N+M) \log \log M\right)$	CREW	
Sorting	$O(\log N)$	$O(N \log N)$	EREW	$A = \Omega(N), T = \Omega(\sqrt{N})$ [1]
Dense Connected Components	$O(\log^2 N)$	$O(N^2)$	Common CRCW	$AT = \Omega(N^3), T = \Omega(N)$ [2]
Sparse Connected Components	$O(\log N)$	$O((M+N) \log N)$	Arbitrary CRCW	$AT = \Omega(N^3), T = \Omega(N)$ [2]
MST	$O(\log^2 N)$	$O(N^2)$	CREW	$AT = \Omega(N^3), T = \Omega(N)$ (by conn. comp.)
Transitive Closure	$O(\log^2 N)$	$O(f(N) \log N)$	CREW	$AT = \Omega(N^3), T = \Omega(N)$ [2]
Optimal Pattern Analysis	$O(\log m)$	$O(m)$	Common CRCW	$AT^2 = \Omega(N^2)$ [7][10]
Suffix Tree Construction	$O(\log N)$	$O(N \log N)$	Arbitrary CRCW	
On-line String Matching	$O(\log m)$	$O(m)$	Arbitrary CRCW	$AT^2 = \Omega(Nm)$ [10]
Longest Repeated Substring	$O(\log \log N)$	$O(N)$	Common CRCW	$AT^2 = \Omega(N^2)$ (by string matching)
FFT	$O(\log N)$	$O(N \log N)$	EREW	$AT^\alpha = \Omega(N^{1+\alpha/2})$ for $0 \leq \alpha \leq 2$ [5]
Polynomial Multiplication	$O(\log N)$	$O(N \log N)$	EREW	$AT^2 = \Omega(N^2)$ (by int. mul.)
Polynomial Division	$O(\log^2 N)$	$O(N \log N)$	EREW	$AT^2 = \Omega(N^2)$ (by int. div.)
Matrix Multiplication	$O(\log N)$ with $p = O\left(\frac{f(N)}{\log N}\right)$			$A = \Omega(N^2), T = \Omega(N)$ [1]
N-bit Multiplication				$AT^\alpha = \Omega(N^{1+\alpha})$ for any $0 \leq \alpha \leq 1$ [4]
Barrel Shift				$A = \Omega(N), T = \Omega(\sqrt{N})$ (transitive)
Discrete Fourier T.				$AT^\alpha = \Omega(N^{1+\alpha/2})$ for $0 \leq \alpha \leq 2$ [5]
Matrix rank				$AT^{2\alpha} = \Omega([kN^2]^{1+\alpha})$ for $0 \leq \alpha \leq 1$ [6]
Determinate Computation				$AT^{2\alpha} = \Omega([kN^2]^{1+\alpha})$ for $0 \leq \alpha \leq 1$ [6]
Singularity Testing				$AT^{2\alpha} = \Omega([kN^2]^{1+\alpha})$ for $0 \leq \alpha \leq 1$ [6]
QR, SVD, LU Decomposition				$AT^{2\alpha} = \Omega([kN^2]^{1+\alpha})$ for $0 \leq \alpha \leq 1$ [6]
N-bit Equality Testing				$AT^2 = \Omega(N^2)$ [7]
Factorization Check ($xy = z$?)				$AT^2 = \Omega(N^2)$ [7]
Matrix-Vector Multiply				$AT^2 = \Omega(N^3)$ [8]
Square Root				$AT^2 = \Omega(N^2)$
Division				$AT^2 = \Omega(N^2)$

Table 1: Some PRAM algorithms that are amenable to translation into AT^2 Interconnect by our method (i.e., $p = O(N)$). $f(N)$ is the number of operations required to perform an $N \times N$ matrix multiplication. p is the number of processors used. N is the problem size. m is the length of the pattern string in string algorithms. h is the depth of the deepest tree in the forest. For matrix multiplication, N is the number of bits required to represent one matrix.

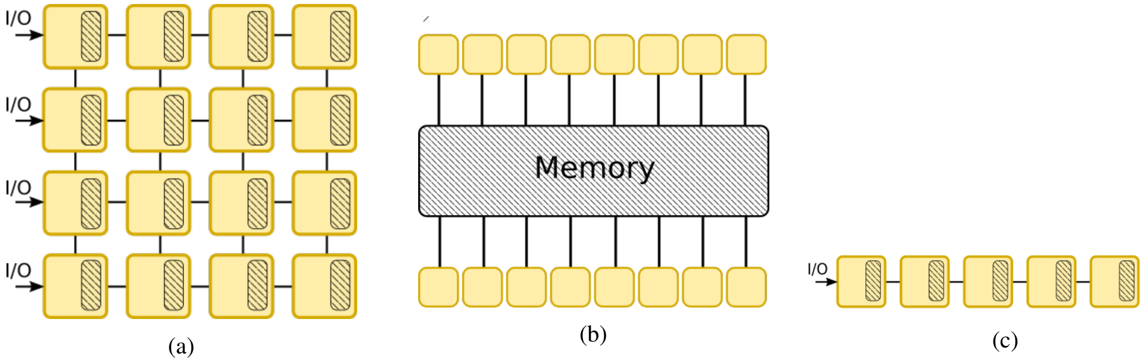


Figure 1: (a) Systolic Array used for translating PRAM to AT^2 Interconnect. The global memory is distributed into the compute cells (b) Illustration of the PRAM model (c) The 1-D Systolic Array used for translating PRAM to Dissipative AT^2 Interconnect.

2 Our Results

2.1 Proposed Systolic Array Architecture

All translations from the PRAM algorithms will be performed using a systolic array architecture (Figure 1c). The N processors are arranged in a square grid with $\sqrt{N}$ processor rows and $\sqrt{N}$ processor columns. Each processor occupies area $O(\log N)$. The processors are connected to adjacent processors with $O(\log N)$ wires. The distance between adjacent processors is constant. The I/O ports are connected to the processors in the first column of the systolic array, with each processor in the first column connecting to $O(\log N)$ I/O ports. We use the same number of input and output ports.

As defined in the AT^2 Interconnect model, the data is loaded and stored in a *when-* and *where-determinate* fashion, meaning that for a fixed problem size N, the bits are loaded in and stored back to off-chip memory in a predetermined fashion no matter the input value.

2.2 Mapping PRAM Processor and Memory to the Systolic Array

Global memory from the PRAM model is mapped onto the systolic array processors. Each processor can hold $O(\log N)$ bits of data. Each processor from the PRAM model is mapped to a single processor on the systolic array, such that the data that under the PRAM model was at address i is now stored in a register in processor i (we use a row-major indexing of processors). Thus, we must have that the number of processor $p \leq N$.

Lemma 2.1. *An arbitrary point within a convex polygon with perimeter N will have distance $\Omega(N)$ to at least one of the vertices of the polygon.*

Lemma 2.2. *From any set of N registers, there exist two registers such that the time required to send a bit between them is $\Omega(\sqrt{N})$. Thus, storing a bit out of a set of N registers into some other register takes time $\Omega(\sqrt{N})$.*

Proof. By the isometric inequality, a convex hull of the N registers will have perimeter $\Omega(\sqrt{N})$. Thus, by Lemma 2.1 we have that there exist two registers that are $\Omega(\sqrt{N})$ apart. □

Theorem 2.1. *With the systolic architecture described above, the N processors can perform an exclusive read of any of the N bits stored in the processors of the systolic array in $O(\sqrt{N})$ time. Also, this result is the best possible.*

Proof. First, by Lemma 2.2 we have that the exclusive reads will take $\Omega(\sqrt{N})$ time between at least a pair of processors.

The challenge now is how to orchestrate N processors to read data between each other on the systolic array with the same asymptotic performance required for a single processor to access one bit out of a set of N bits ($O(\sqrt{N})$). We propose the following scheme:

1. Each processor i determines a destination address it wants to read (d_i). Note that we assume the reads are exclusive, meaning no two processors read the same bit at the same time under the PRAM model. The concurrent case is covered in Theorem 2.2.

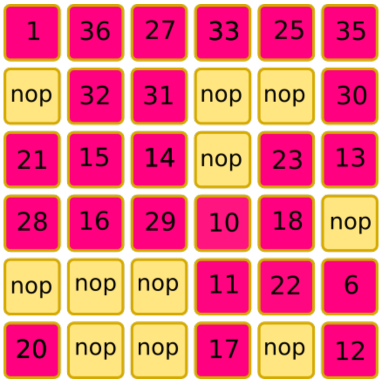


Figure 2: An example configuration of read requests (destination addresses written on each processor)

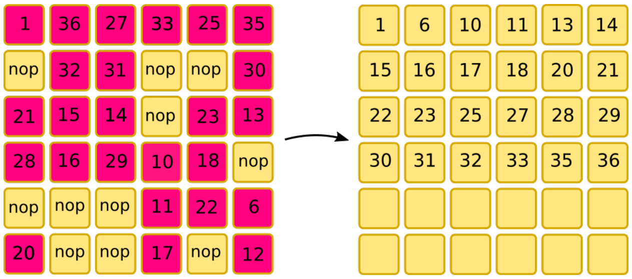


Figure 3: Illustration of step 3 of the algorithm in Theorem 2.1 (destination addresses written on each processor)

2. Each processor i stores a tuple (i, d_i) in a special *communication register*. Some processors may also idle for a cycle under the PRAM model and do not issue reads. In that case, they store a special `nop` signal in their communication register, which is considered bigger than all other tuples. ($O(1)$ time; Figure 2)
3. The bitonic mesh sorting algorithm sorts the tuples into row-major increasing order by the destination address d_i. ($O(\sqrt{N})$ time; Figure 3)
4. Each tuple (i, d_i) is distributed along its current column to the row in which the processor with index d_i is located in. ($O(\sqrt{N})$ time; Figure 4)
5. Each tuple (i, d_i) is transported along its current row to processor d_i. ($O(\sqrt{N})$ time; Figure 5)
6. For all i, processor d_i stores the content of its memory register (c_i) into a tuple (i, c_i). ($O(1)$ time)
7. The tuples (i, c_i) are distributed back to their original processor i using the same scheme as was used to distribute read requests to appropriate processors. ($O(\sqrt{N})$ time)

At this point, each processor i has access to the contents c_i stored at address d_i. It is important to note that in step 4 can be done. That is, that for a given row, there can be at most one tuple per column whose destination processor is in that

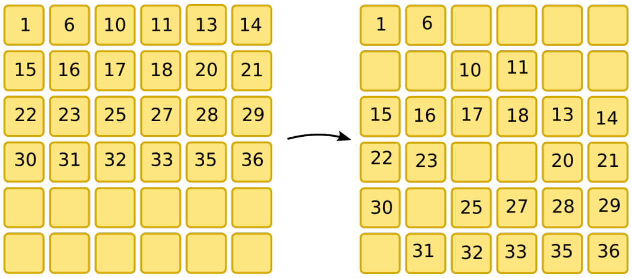


Figure 4: Illustration of step 4 of the algorithm in Theorem 2.1 (destination addresses written on each processor)

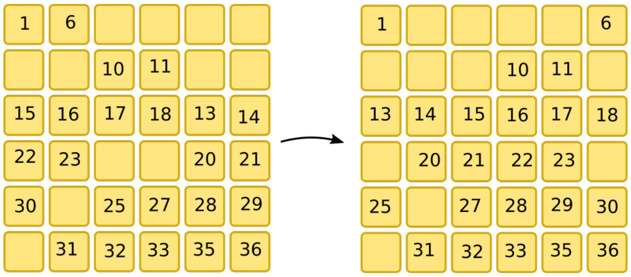


Figure 5: Illustration of step 5 of the algorithm in Theorem 2.1 (destination addresses written on each processor)

row. This is guaranteed by sorting in step 3. Namely, looking at any two tuples that are north-south adjacent, since all d_i's are distinct due to the exclusive read requirement, we have that there are at least $\sqrt{N}$ d_i's that are strictly between them (all the tuples that are stored to the right of the north processor and to the left of the south processor). This implies that the difference between the north and south d_i's is at least $\sqrt{N}$. Thus, in a row-major indexing, they cannot be in the same row as all processors in the same row have indices that are within $\sqrt{N}$ of each other.

Further, notice that step d_i can be done with $O(\log N)$ registers in each processors, as at most three elements can intersect at any given time during their transport to the appropriate processor. Specifically, in the worst case three tuples collide in one processors when one processor is going right, one is going left, and one that has already reached its destination in the current processor. □

Now we extend this result to concurrent reads.

Theorem 2.2. *With the systolic architecture described above, the N processors can perform a concurrent read of any of the N bits stored in the processors of the systolic array in $O(\sqrt{N})$ time. Also, this result is the best possible.*

Proof. We will adapt the algorithm described in Theorem 2.1 to handle multiple processors reading from a single destination address. The steps $1-3$ are the same as in Theorem 2.1. After the sort, all processors except the ones in the last (right-most) column send their current value to their right neighbor. The processors in the last (right-most) column send their current d_i value to the next processor in the row-major ordering (this processor is in the first column, one row under the processor). It takes $O(\sqrt{N})$ time for this signal to propagate. When the processors receive their respective signals, they check if the received value is the same as their d_i value. If it is, then they are storing a duplicate destination address and do not do anything. If the received value is different, then the processor is the first (in the row-major ordering) to store d_i and asserts a special *first* flag and stores the tuple (l_i, d_i) in its *first tuple* register, where l_i is the row-major index of the current processor in which d_i is stored.

Now, treating the first tuple registers as communication registers, we perform the algorithm described in Theorem 2.1. When the algorithm finishes, the first registers of the processors with the first flag asserted will hold the contents of the appropriate destination address. These contents c_i now need to be flooded to all other processors that hold the same destination address d_i. Since the d_i's have been initially sorted, the flooding can be done from the processors with the first flag asserted to adjacent processors that have the same d_i. This can be done in time $O(\sqrt{N})$ by sending the (c_i, d_i) values to all adjacent processors and having processors accept a c_i if the d_i matches their d_i and then they continue this flooding procedure to further processors. When the data is flooded, the elements are sorted back based on their source index just as in step 7 of Theorem 2.1. □

Theorem 2.3. *With the systolic architecture described above, the N processors can perform an exclusive write of any of the N bits stored in the processors of the systolic array in $O(\sqrt{N})$ time. Also, this result is the best possible.*

Proof. To do an exclusive write, we perform the steps 1-5 in Theorem 2.1, but we use a tuple (d_i, w) where d_i is the write destination address and w is the data to be written. When the tuple holding d_i is at processor d_i we can write w into processor d_i and the procedure is finished. □

Theorem 2.4. *With the systolic architecture described above, the N processors can perform a concurrent write of any of the N bits stored in the processors of the systolic array in $O(\sqrt{N})$ time. Also, this result is the best possible.*

Proof. We use tuples (d_i, w) as in Theorem 2.3 and perform the steps 1-3 in Theorem 2.1. Once the data is sorted by destination address, adjacent fields that collide will be adjacent. At this point an appropriate protocol is run to reach consensus between the tuples that have the same destination address on what happens next. For example, when a priority concurrent write is used, each processor is assigned a unique priority and the processor with the highest priority on a write collision will commit its w value. The tuple with the highest priority can be identified between adjacent processors in $O(\sqrt{N})$ time by a similar flooding procedure as described in Theorem 2.2 (the priority of the processor would be included in the tuple). Once the maximum priority tuple is established, it sends its w value to the lowest index with the same destination address (again, using the same flooding algorithm). Once the processors holding the lowest index among the ones with the same destination address receive the highest priority write value w, we assert the first flag of these lowest index processors, populate the first register with the tuple (d_i, w), and perform steps $3 - 5$ until the tuple (d_i, w) is in processor d_i, for all i. Then commit the value w to processor d_i, for all i.

Alternatively, if a common concurrent write procedure is used, a write is committed only if all the colliding w values are equal. Similar to the previous approach, the adjacent w values are flooded and compared. Then the first flag is asserted in the lowest index processor if all the w values that collide with it are equal, otherwise not. Then we proceed as in the priority concurrent case after the first flag is asserted. □

Theorem 2.5. *Each EREW, CREW, or CRCW p-processor PRAM algorithm with $T = O(\log N)$ and $p = O(N)$ can be mapped to the systolic array with multiplicative overhead in execution time of $O(\sqrt{N})$ under the AT^2 Interconnect metric. This result is the best possible.*

Proof. We load the data through the I/O ports in a systolic fashion, with the contents of the first $\sqrt{N}$ addresses loaded through the top-most I/O port. The element in the first row is transposed to the right one cycle at a time until all the rows are filled. In general, the i-th row is filled with elements with addresses in the range $((i-1)\sqrt{N}, i\sqrt{N}]$. This can be done in $O(\sqrt{N})$ time using $\sqrt{N}$ ports and achieves a row-major addressing scheme.

Each cycle of a PRAM algorithm has three parts. First, the processor request data from memory, then the data is loaded, and finally, processor write some data to memory. In the PRAM model, these three steps are considered to run in $O(1)$. However, as we have shown in Theorems 2.1, 2.3, 2.2, and 2.4, in general the runtime of the reads and writes will be $\Theta(\sqrt{N})$ under the AT^2 Interconnect model. Thus, our read and write algorithms give a mapping from PRAM algorithms to the systolic array such that the impact of runtime is $O(\sqrt{N})$. This result is tight, because each processor makes $T(N)$ accesses, each of which require time $O(\sqrt{N})$ as proven in Lemma 2.2 (here, $T(N)$ denotes the execution time of the PRAM algorithm on inputs of size N). □

Remark 2.1. *The key insight here is that by looking at Table 1, most listed problems have AT^2 Interconnect complexity of $A = \Omega(N), T = \Omega(\sqrt{N})$. Further, all the PRAM algos use $\leq N$ processors and require $O(\log^k N)$ time. Thus, Theorem 2.5 gives a methodical way to translate PRAM-optimal designs into a single piece of hardware (Systolic Array) which will have optimal area and optimal time (up to logarithmic factors).*

Although the result of Theorem 2.5 is tight for general PRAM algorithms, better results can be achieved if we can place additional assumptions on the access pattern locality of each processor. We now define a subset of PRAM algorithms that lends itself to better translations into the AT^2 Interconnect universe. Although the definition seems specific, it covers many of the most useful algorithms.

Definition 2.1. *A PRAM algorithm is said to have an amenable access pattern if it runs in $O(\log N)$ time and for all $d \leq \log N$, we have that in the first $O(2^d)$ cycles, processors $[i2^d, (i+1)2^d)$ access only the addresses in the range $[i2^d, (i+1)2^d)$.*

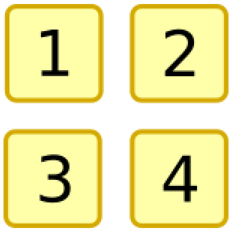
You need to talk about which algorithms have an amenable access pattern here.

Definition 2.2. *For any $d \in N$, we define a binary tree mapping from the set $\{1, 2, ..., 2^{4d}\}$ onto the systolic area of dimension $2^{2d} \times 2^{2d}$ recursively:*

1. *If $d = 0$, then put 1 into the only available square.*
2. *Otherwise, take the $2^{2d-2} \times 2^{2d-2}$ binary tree mapping and put it into the top left corner. Then add 2^{4d-4} to each element in the $2^{2d-2} \times 2^{2d-2}$ mapping and put the resulting mapping into the top right corner. Again, add 2^{4d-4} and put the resulting mapping to the bottom left corner. Add 2^{4d-4} again and put the result into the bottom right corner.*

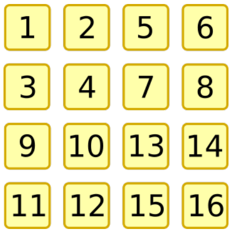
Lemma 2.3. *For all $d \leq D$, the binary tree mapping of size $2^{2D} \times 2^{2D}$ has the maximum distance between any two processors with labels in the range $[i2^d, (i+1)2^d)$ is $2 \cdot 2^d$.*

Proof. The proof follows from the inductive construction of the mapping. □



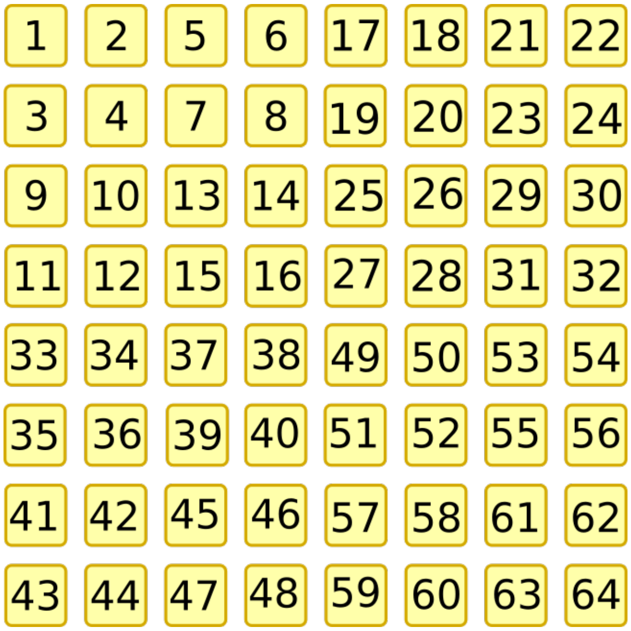
1	2
3	4

(a) Illustration of a binary tree mapping onto an 2×2 grid



1	2	5	6
3	4	7	8
9	10	13	14
11	12	15	16

(b) Illustration of a binary tree mapping onto an 4×4 grid



1	2	5	6	17	18	21	22
3	4	7	8	19	20	23	24
9	10	13	14	25	26	29	30
11	12	15	16	27	28	31	32
33	34	37	38	49	50	53	54
35	36	39	40	51	52	55	56
41	42	45	46	57	58	61	62
43	44	47	48	59	60	63	64

(c) Illustration of a binary tree mapping onto an 8×8 grid

Theorem 2.6. *If its access pattern is amenable, a PRAM algorithm can be mapped to the systolic array with $O(\sqrt{N})$ additive overhead under the AT^2 Interconnect metric.*

Proof. As shown in Lemma 2.3, the total overhead of the read/write procedure in the first $O(\sqrt{2^d})$ cycles equals $O(2^d)$ plus the overhead of the first $O(\sqrt{2^{d-1}})$ cycles, with the overhead for $d = 1$ being $O(1)$. Thus, the total overhead of the reads and writes on the runtime is $O\left(\sum_{i=0}^{\log N} \sqrt{2^i}\right) = O(\sqrt{N})$ additive. □

2.3 Dissipative Model

Theorem 2.7. *Under the dissipative model, N processors can do an exclusive read of N bits in $O(N)$ time. This result is the best possible. [Basically this is a similar idea as before, except we use a $1 \times N$ systolic array with N ports instead of a $\sqrt{N} \times \sqrt{N}$ systolic array with $\sqrt{N}$ ports. I will include the details later, but the flavor and significance of the result is similar to what was discussed for AT^2 Interconnect.]*

References

[1] Bernard Chazelle, Louis Monier. *A Model of Computation for VLSI with Related Complexity Results*. Journal of the Association for Computing Machinery, Volume 32, No. 3, July 1985, pp. 573-588.

[2] Joseph JaJa. *The VLSI Complexity of Selected Graph Problems*. Journal of the ACM, Volume 31 Issue 2, April 1984. pp. 377-391

[3] Joseph JaJa. *Introduction to Parallel Algorithms.*

[4] R. P. Brent, H. T. Kung. *The Chip Complexity of Binary Arithmetic.* Symposium on the Theory of Computing, 1980, pp. 190-200.

[5] C. D. Thompson, *Area-Time Complexity for VLSI.* Symposium on The Theory of Computing, 1979, pp. 81-88.

[6] Jeff I. Chu, Georg Schnitger. *The Communication Complexity of Several Problems in Matrix Multiplication.* Journal of Complexity, 1991, pp. 395-407.

[7] Richard J. Lipton, *Lower Bounds for VLSI*. Symposium on the Theory of Computing, 1981, pp. 300-307.

[8] B. Codenotti, G. Lotti, F. Romani. *Area-Time Trade-Offs for Matrix-Vector Multiplication.* Journal of Parallel and Distributed Computing, 1990, pp. 52-59.

[9] Kurt Mehlhorn. AT^2*-Optimal VLSI Integer Division and Integer Square Rooting.* Integration Letters, 1984.

[10] Micah Adler, John Byers, AT^2 *Bounds for a Class of VLSI Problems and String Matching*.

[11] Yi-Hsiang Lai, Yuze Chi, Yuwei Hu, Jie Wang, Cody Hao Yu, Yuan Zhou, Jason Cong, Zhiru Zhang. *HeteroCL: A Multi-Paradigm Programming Infrastructure for Software-Defined Reconfigurable Computing*. International Symposium on Fild-Programmable Gate Arrays (FPGA), February, 2019. (Best Paper Award)

[12] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, Saman Amarasighe. *Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines.* Conference on Programming Language Design and Implementation, 2013.

[13] Xuan Yang, Mingyu Gao, Jing Pu, Ankita Nayak, Qiaoyi Liu, Steven Emberton Bell, Jeff Ou Setter, Kaidi Cao, Heonjae Ha, Christos Kozyrakis, and Mark Horowitz. *DNN Dataflow Choice is Overrated.* arXiv:1809.04070

[14] Jason Cong, Bin Liu. *A Metric for Layout-Friendly Microarchitecture Optimization in High-Level Synthesis*. Design Automation Conference, 2012.

[15] Richard Cole. *Parallel Merge Sort*. Society of Industrial and Applied Mathematics Journal on Computing, Volume 17 Issue 4, 1988, pp. 770-785.

[16] Jia-Wei Hong, H.T. Kung. *I/O Complexity: The Red-Blue Pebble Game*. Symposium on the Theory of Computation, 1981, pp. 326-333.

[17] Dror Irony, Sivan Toledo, Alexander Tiskin. *Communication Lower Bounds for Distributed-Memory Matrix Multiplication*. Journal of Parallel and Distributed Computing, Volume 64, Issue 9, September 2004, pp. 1017-1026.

[18] Thomas Lengauer, Kurt Mehlhorn. *VLSI Complexity, Efficient VLSI Algorithms, and the HILL Design System*. Proceedings of the International Professorship, 1983.

[19] Bernard Chazelle, Louis Monier. *A Model of Computation for VLSI with Related Complexity Results*. Journal of the Association for Computing Machinery, Volume 32, No. 3, July 1985, pp. 573-588.

[20] Venmugil Elango, Emre Kurt, Julien Langou, Louis-Noel Pouchet, Fabrice Rastello, Aravind Sukumaran-Rajam, Saday Sadayappan. *Characterization of the Data Movement Complexity of Computations.*

[21] Alok Aggarwal, Ashok K. Chandra, Marc Snir. *Communication Complexity of PRAMs*. Journal of Theoretical Computer Science, Volume 71, Issue 1, 1990, pp. 3-28.

[22] Peipei Zhou, Hyunseok Park, Zhenman Fang, Jason Cong, Andre DeHon. *Energy Efficiency of Full Pipelining: A Case Study for Matrix Multiplication*. International Symposium on Field-Programmable Custom Computing Machines, 2016.

[23] Joseph JaJa. *Introduction to Parallel Algorithms*. 1992.

[24] Clark D. Thompson. *The VLSI Complexity of Sorting*. IEEE Transactions on Computers, 1983.

[25] David Nassimi, Sartaj Sahni. *Bitonic sort on a mesh-connected parallel computer*. IEEE Transactions on Computers, 1979.

[26] Jean Vuillemin. *A Combinatorial Limit to the Computing Power of VLSI Circuits.* IEEE Transactions on Computers, 1983, pp. 294-300.

[27] C.D. Thompson, H.T. Kung, *Sorting on a Mesh-Connected Parallel Computer*. Communications of the ACM, 1977.

[28] Alok Aggarwal, Jeffrey Scott Vitter. *The Input/Output Complexity of Sorting and Related Problems*. Communications of the ACM, Volume 31, Number 9, September 1988.

[29] Nikola Samardzic, Weikang Qiao, Vaibhav Aggarwal, M.C. Frank Chang, Jason Cong. *High-Performance Adaptive Merge Tree Sorting*.

[30] Ana Klimovic, Heiner Litz, Christos Kozyrakis. *ReFlex: Remote Flash ≈ Local Flash.* International Conference on Architectural Support for Programming Languages and Operating Systems.

[31] Tyler Michael Smith, Bradley Lowery, Julien Langou, Robert A. van de Geijn. *A Tight I/O Lower Bound for Matrix Multiplication*. arXiV, 2019.

[32] https://mail.google.com/mail/u/0/inbox `https://www.xilinx.com/support/documentation/white_papers/wp477-ultraram.pdf`

[33] Jason Cong, Jie Wang. *PolySA: Polyhedral-based Systolic Array Auto-compilation*. International Conference on Computer-Aided Design, 2018.

[34] H. T. Kung. *Why Systolic Architectures?* Computer, Vol. 15, Iss. 1, January 1982.

[35] Sanjeev Arora, Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press.

[36] Andrew Chi-Chih Yao. *Some Complexity Questions Related to Distributive Computing*. Symposium on the Theory of Computing, 1979.

[37] Luiz André Barroso, Jimmy Clidaras, Urs Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool Publishers.

[38] Arvind Krishnamurthy. *PRAM Algorithms*. University of Washington CS 424, Fall 2004.

[39] J. E. Savage. *Area-time tradeoffs for matrix multiplication and related problems in VLSI models*. Journal of Computing Systems Science, 1981, pp. 230-242.

[40] Jeffrey D. Ullman. *Computational Aspects of VLSI*. Computer Science Press, 1983.

[41] Stephen Cook, Dwork, and Reischuk. *Upper and lower bounds for parallel random access machines without simultaneous writes*. preprint, 1983.