Adding and Multiplying in the Tile Assembly Model

Yuriy Brun

Department of Computer Science
University of Southern California
Los Angeles, CA 90089
ybrun@usc.edu

Abstract. The tile assembly model, a formal model of crystal growth, is of special interest to computer scientists and mathematicians because it is universal [1]. Therefore, tile assembly model systems can compute all the functions that computers compute. In this paper, I formally define what it means for a system to compute a function deterministically and present systems that add and multiply. While the proof that the tile assembly model is universal [2] implies the construction of such systems, those systems are in some sense "large" and "slow." The systems presented here all use $\Theta(1)$ different tiles (8 to add and 28 to multiply) and compute in time linear in the input size.

1 Introduction

Self-assembly is a process that is ubiquitous in nature. Systems form on all scales via self-assembly: atoms self-assemble to form molecules, molecules to form complexes, and stars and planets to form galaxies. One manifestation of self-assembly is crystal growth: molecules self-assembling to form crystals. Crystal growth is an interesting area of research for computer scientists because it has been shown that, in theory, under careful control, crystals can compute [2].

While DNA computation suffers from relatively high error rates, the study of self-assembly shows how to utilize redundancy to design systems with built-in error correction [3–5]. Barish et al. have demonstrated a DNA implementation of tile systems, one that copies an input and another that counts in binary [6]. Similarly, Rothemund et al. have demonstrated a DNA implementation of a tile system that computes the *xor* function, resulting in a Sierpinski triangle [7].

1.1 Tile Assembly Model

The tile assembly model [1, 2, 8] is a formal model of crystal growth. It was designed to model self-assembly of molecules such as DNA. It is an extension of a model proposed by Wang [9].

The model has square *tiles* that may stick together based on various *binding domains* on their four sides. The four binding domains, elements of a finite alphabet Σ, define the type of the tile. The strength of the binding domains are defined by the *strength function* g. The placement of some tiles on a 2-D grid is called a *configuration*, and a tile may *attach* in empty positions on the grid if the total strength of all the binding domains on that tile that match its neighbors exceeds the current *temperature* (a natural number). Finally, a *tile system* $\mathbb{S}$ is a triple $\langle T, g, \tau \rangle$, where T is a finite set of tiles, g is a strength function, and $\tau \in \mathbb{N}$ is the temperature, where $\mathbb{N} = \mathbb{Z}_{\geq 0}$.

Starting from a *seed configuration* S, tiles may attach to form new configurations. If that process terminates, the resulting configuration is said to be *final*. At some times, it may be possible for more than one tile to attach at a given position, or there may be more than one position where a tile can attach. If for all sequences of tile attachments, all possible final configurations are identical, then $\mathbb{S}$ is said to produce a *unique* final configuration on S. The *assembly time* of the system is the minimal number of steps it takes to build a final configuration, assuming maximum parallelism.

Let f be a function $f : \mathbb{N}^n \to \mathbb{N}^m$. A tile system $\mathbb{S}$ is said to *deterministically compute* f if these exists a seed that encodes $\boldsymbol{a} \in \mathbb{N}^n$ and $\mathbb{S}$ produces a unique final configurations F that encodes $f(\boldsymbol{a})$.

I refer the reader to [10] for more formal versions of the definitions. In the remainder of this paper, I will examine adders, systems that compute $f(\alpha, \beta) = \alpha + \beta$ and multipliers, systems that compute $f(\alpha, \beta) = \alpha\beta$. I require systems to encode their inputs in binary, and call the set of tiles used to encode the input Γ.

1.2 Adding

I present a system that uses eight tiles to add numbers. Intuitively, $\mathbb{S}_{+8}$ has eight tiles with the east, north, and south sides as the input sides and the west side as the output side, outputting 1 iff the sum of the inputs is at least 2. To encode the answer, the type of the tile is determined by the sum of the inputs, modulo 2. Figure 1(a) shows the eight tiles for all possible 0 and 1 binding domains for the three input sides. The 1 or 0 in the middle of each tile is the output bit that that tile encodes (the tile's v_8 value).

Theorem 1. *Let $\Sigma_8 = \{0, 1\}$, $g_8 = 1$, $\tau_8 = 3$, and T_8 be a set of tiles over Σ_8 as described in Figure 1(a). Then $\mathbb{S}_{+8} = \langle T_8, g_8, \tau_8 \rangle$ computes the function $f(\alpha, \beta) = \alpha + \beta$.*

Figure 1(b) shows a sample seed configuration which encodes two numbers in binary: $100010_2 = 34$, $11011_2 = 27$. Number 34 is encoded on the top row and number 27 is encoded on the bottom row. There are 5 tiles in Γ_8, the 1 and 0 tiles for each of the two inputs, and the single starter tile on the right side. Note that at the start, only one tile may attach to this configuration because $\tau_8 = 3$. Figure 1(c) shows the final configuration for the example of $34 + 27$, with the solution encoded on the center row. The row reads 111101_2 which is $61 = 34 + 27$. Because the sum of two n-bit numbers may be as large as $n + 1$

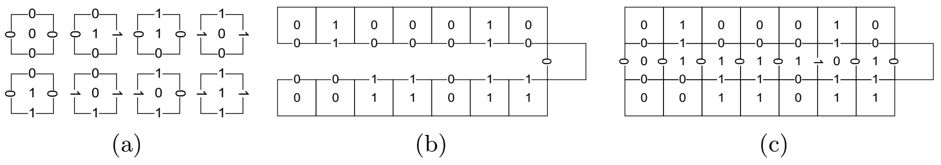


Fig. 1. Tile system $\mathbb{S}_{+8}$ computes the function $f(\alpha,\beta) = \alpha + \beta$. Each of the eight tiles (a) has 3 input sides (east, north, and south) and 1 output side (west) and is labeled with a 1 or a 0 to assist the reader with reading the encoding v_8. Given a sample input of $\beta = 100010_2 = 34$ and $\alpha = 11011_2 = 27$ (b), with 34 on top row and 27 on the bottom row, the system fills the row in the middle with $\alpha + \beta = 111101_2 = 61$ to produce the unique final configuration (c). Note that the least significant digit is on the right. Also note that the inputs are padded with extra 0 tiles in the most significant bit places because the sum of two n-bit numbers may be as large as $n + 1$ bits.

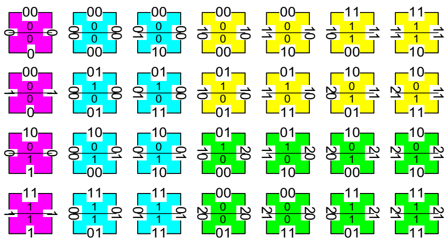
bits, each of the two inputs needs to be padded to be $n + 1$ bits long with extra 0 tiles.

The logic of the system is identical to a series of 1-bit full adders. Each solution tile takes in a bit from each of the inputs on the north and south sides and a carry bit from the previous solution tile on the east side, and outputs the next carry bit on the west side. Because $\tau_8 = 3$, only a tile with three neighbors may attach at any time, and therefore, no tile may attach until its right neighbor has. Thus the assembly time for this system is n steps to add two n-bit numbers. Note that $|\Gamma_8| = 5$ and $|T_8| = 8$. The complete proof of theorem 1, and of the assembly time of $\mathbb{S}_{+8}$ can be found in [10].

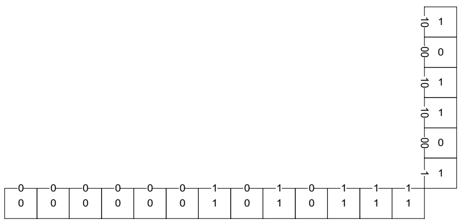
1.3 Multiplying

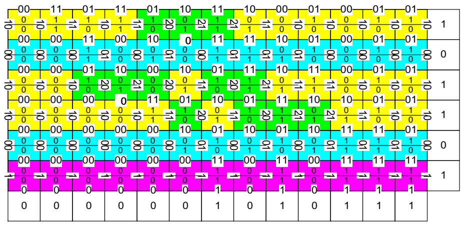
While I have only described a single adder system, there are many other such systems. Some of them add a single bit per row. While not very interesting as an adder system on its own, such a system will be very helpful in making a multiplier system. I do not formally define such an adder here, but there are two instances of such adders in [10]. I now present a multiplier system that performs part of the computation on each row, and presents the product of two inputs on the top row of an almost complete rectangle.

Observe that if $\beta = \sum_i \beta_i 2^i$, for $\beta_i \in \{0,1\}$, that is, the i^{th} bit of β is β_i, then the product $\alpha\beta$ can be written as $\alpha\beta = \sum_i \beta_i \alpha 2^i$. That is, one can multiply α by each of the powers of 2 in β, and then add up the products. Multiplying by 2 in binary is simple — it is a left-shift. A system with tiles that "flow" the information to the left and display the information received from the right would perform such a left-shift. What is left is to add the rows representing the appropriate powers of 2 to construct the product of two numbers. The system should, therefore, add up to n numbers, one per row. It is feasible to imagine such a system that on each row adds a new number to a running total, and arrives at the sum of n numbers on the n^{th} row. Thus I have informally described two



(a)





(b) (c)

Fig. 2. Tile system $\mathbb{S}_\times$ computes the function $f(\alpha, \beta) = \alpha\beta$ and uses 28 distinct tile types (a). Given a sample input of $\alpha = 1010111_2 = 87$, and $\beta = 101101_2 = 45$ (b), with 87 on bottom row and 45 on the right-most column, the system fills the rectangle (c). The i^{th} row has two pieces of information: on the lower half of each tile is the value of the appropriate bit of $2^i\alpha$ and on the upper half is the running sum of the products of α and powers of 2 in β smaller or equals to i. The upper portion of the top row reads the solution: $\alpha\beta = 0111101001011_2 = 3915 = 87 \cdot 45$.

sets of tiles: one that performs a left-shift and one that adds a number to the running total on each row. Combining the functionality of these tiles into one system produces a multiplier system.

Figure 2(a) shows the tiles of $\mathbb{S}_\times$ and Figures 2(b) and 2(c) show an example execution of $\mathbb{S}_\times$ on $\alpha = 87$ and $\beta = 45$. The input α is encoded on the bottom row and the input β is encoded on the right-most column. There are 4 special magenta tiles which deal with the input and fill in the bottom row of the computation. These tiles are necessary because the least significant bit of the column input is 2^0 and thus requires no left-shift. The blue tiles code rows that have a 0 in the input β. These tiles perform a left-shift, but do not add the current power of 2 to the running sum. The green and yellow tiles fill in the rows that have a 1 in the input β. Yellow tiles indicate the incoming carry bit is 0, and green tiles indicate that the bit is 1. Each tile, in addition to its binding domains, is labeled with two pieces of information: the lower half of the tile on the i^{th} row displays the appropriate bit of $2^i a$ and the upper half of the tile displays the appropriate bit of the running sum so far. The top half of the top row displays the solution. In the example, the solution is $0111101001011_2 = 3915 = \alpha\beta$.

Theorem 2. *Let* $\Sigma_\times = \{0, 1, 00, 01, 10, 11, 20, 21\}$*,* $g_\times = 1$*,* $\tau_\times = 2$*, and* $T_\times$ *be a set of tiles over* $\Sigma_\times$ *as described in Figure 2(a). Then* $\mathbb{S}_\times = \langle T_\times, g_\times, \tau_\times \rangle$ *computes the function* $f(\alpha, \beta) = \alpha\beta$*.*

There are 6 tiles in $\Gamma_\times$, the 1 and 0 tiles for each of the inputs and special 0 and 1 tiles for the least significant bit of β. Because the product of two n-bit numbers may be as large as $2n$ bits, the row input to the multiplier system needs to be padded with n extra 0 tiles. The column input does not need this padding. The assembly time for this system is $\Theta(n_\alpha + n_\beta)$. Note that $|\Gamma_\times| = 6$ and $|T_\times| = 28$. The complete proof of theorem 2, and of the assembly time of $\mathbb{S}_\times$ can be found in [10].

2 Contributions

The tile assembly model is a formal model of self-assembly and crystal growth. I explored what it means for a tile assembly system to compute a function deterministically and designed systems to add and multiply. The adding system uses 8 and the multiplying system uses 28 computational tiles. Both systems use $\Theta(1)$ input tiles and compute in $\Theta(n)$ steps for an n-bit input.

References

1. Winfree, E.: Simulations of computing by self-assembly of DNA. Technical Report CS-TR:1998:22, California Insitute of Technology, Pasadena, CA (1998)
2. Winfree, E.: Algorithmic Self-Assembly of DNA. PhD thesis, California Insitute of Technology, Pasadena, CA (June 1998)
3. Winfree, E., Bekbolatov, R.: Proofreading tile sets: Error correction for algorithmic self-assembly. In: The 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS02). Volume 2943., Madison, WI (June 2003) 126–144
4. Chen, H.L., Goel, A.: Error free self-assembly with error prone tiles. In: Proceedings of the 10th International Meeting on DNA Based Computers, Milan, Italy (June 2004)
5. Reif, J.H., Sahu, S., Yin, P.: Compact error-resilient computational DNA tiling assemblies. In: Proceedings of the 10th International Meeting on DNA Based Computers, Milan, Italy (June 2004)
6. Barish, R., Rothemund, P., Winfree, E.: Two computational primitives for algorithmic self-assembly: Copying and counting. Nano Letters **5**(12) (2005) 2586–2592
7. Rothemund, P., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biology **2**(12) (2004) e424
8. Rothemund, P., Winfree, E.: The program-size complexity of self-assembled squares. In: ACM Symposium on Theory of Computing (STOC02), Montreal, Quebec, Canada (2001) 459–468
9. Wang, H.: Proving theorems by pattern recognition. I. Bell System Technical Journal **40** (1961) 1–42
10. Brun, Y.: Arithmetic computation in the tile assembly model: Addition and multiplication. Theoretical Computer Science **10.1016/j.tcs.2006.10.025** (2006)