Block-space GPU Mapping for Embedded Sierpiński Gasket Fractals

Cristóbal A. Navarro
Institute of Informatics,
Universidad Austral de Chile,
Valdivia, Chile
Email: cnavarro@inf.uach.cl

Benjamín Bustos
Department of Computer Science (DCC)
University of Chile, Santiago, Chile
Email: bbustos@dcc.uchile.cl

Raimundo Vega
Institute of Informatics,
Universidad Austral de Chile,
Valdivia, Chile
Email: rvega@inf.uach.cl

Nancy Hitschfeld
Department of Computer Science (DCC)
University of Chile, Santiago, Chile
Email: nancy@dcc.uchile.cl

Abstract—**This work studies the problem of GPU thread mapping for a Sierpiński gasket fractal embedded in a discrete Euclidean space of $n \times n$. A block-space map $\lambda : \mathbb{Z}_{\mathbb{E}}^2 \mapsto \mathbb{Z}_{\mathbb{F}}^2$ is proposed, from Euclidean parallel space $\mathbb{E}$ to embedded fractal space $\mathbb{F}$, that maps in $\mathcal{O}(\log_2 \log_2(n))$ time and uses no more than $\mathcal{O}(n^{\mathbb{H}})$ threads with $\mathbb{H} \approx 1.58...$ being the Hausdorff dimension, making it parallel space efficient. When compared to a bounding-box map, $\lambda(\omega)$ offers a sub-exponential improvement in parallel space and a monotonically increasing speedup once $n > n_0$. Experimental performance tests show that in practice $\lambda(\omega)$ can produce performance improvement at any block-size once $n > n_0 = 2^8$, reaching approximately $10\times$ of speedup for $n = 2^{16}$ under optimal block configurations.**

Keywords—*GPU computing; thread mapping; block-space fractal domains; Sierpinski gasket;*

I. INTRODUCTION

Fractals can be described as self-similar structures [14] where a *similar*[1] geometrical pattern is found at all scales. Several natural phenomena produce fractal patterns that obey a self-similar structure [13]. Phenomena such as plant and tree growth [21], [22], terrain formation [15], [23], molecular dynamics [26], snowflake crystallization [9], blood vessels [5], morphological features of living organisms [27], among many others, display a fractal design where self-similarity is a critical feature for modeling its geometrical structure.

One well known fractal is the *Sierpiński Gasket*, or *Sierpiński Triangle*, described by Waclaw Sierpiński in 1915. Despite being over than a century old, the Sierpiński gasket remains a relevant subject as it is present in different fields such as the construction of antennas [2], [24], cellular automata [20], [28], molecular DNA self-organization [10], [26], self-assembly theory [4], [12] and phase transitions on fractal spin lattices [6], [7], [16], among others, because of its special properties. For fractal simulations such as in Cellular Automata or Monte Carlo simulation on spin models, the Sierpiński gasket is usually discretized. Is this form, the fractal is defined as a self-similar structure where level $r + 1$ is composed of

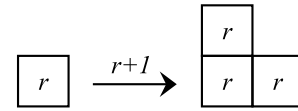three repetitions of level r at $1/2$ the scale, as shown in Figure 1.

Fig. 1: Construction of the discrete Sierpiński gasket.

Applications that involve graphical representations or data-parallel simulations with nearest neighbors interactions may benefit if the storage of the structure preserves spatial locality in memory space, *i.e*, the memory locations $(x \pm 1, y \pm 1)$ define a neighborhood in the actual fractal as well. One way to achieve this is to embed the fractal in a Euclidean space of $n \times n$ as shown in Figure 2.
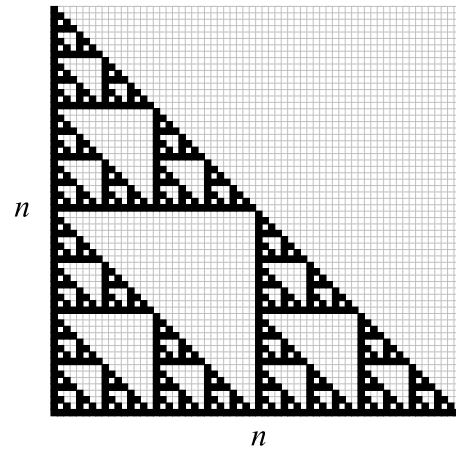

Fig. 2: A discrete Sierpiński gasket embedded in a $n \times n$ Euclidean space.

For applications that can operate in an embedded fractal domain, computations will usually consist of accessing all the

---

[1]Depending on which fractal, the term *similar* can refer to *exactly similar* or *quasi similar*.

elements of the fractal and eventually perform arithmetic/logic operations that involve the data-element itself and possibly its nearest neighbors. Eventually, when the fractal is large enough to the point of containing hundreds of thousands of elements, a sequential computation can take an excessive amount of time for the practical requirements of the field. In these situations GPU computing becomes an attractive tool for accelerating the task [19].

For every GPU computation there is a stage where threads are mapped from parallel to data space. A map, defined as $f : \mathbb{Z}^k \rightarrow \mathbb{Z}^m$, transforms each k-dimensional point $x = (x_1, x_2, ..., x_k)$ in parallel space P^k into a unique m-dimensional point $f(x) = (y_1, y_2, \cdots, y_m)$ in data space D^m. GPU parallel spaces are defined as orthotopes $\Pi^k \in P^k$ in $k = 1, 2, 3$ dimensions. A known way of mapping threads is to use the *bounding-box* (BB) approach, that builds an orthotope Π^k sufficiently large to cover the whole data space and threads are mapped using the identity $f(\omega) = \omega$. Such map is highly convenient and efficient for the class of problems where data space is also defined by an orthotope; such as vectors, tables, matrices and box-shaped volumes. But for an embedded fractal such as the Sierpiński gasket, this approach is no longer efficient in terms of parallel space as many threads fall outside the domain, introducing a performance penalty to the execution time (see Figure 3).
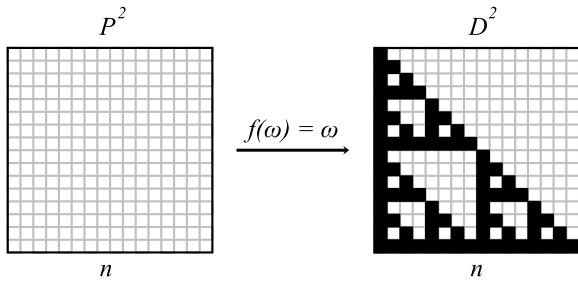

Fig. 3: In the bounding-box approach the threads that fall outside the fractal have to be discarded at run time.

Two research questions arise from this GPU efficiency problem on the embedded Sierpiński gasket; The first question: *Is there any parallel-space efficient function, namely $\lambda(\omega)$, that can use asymptotically the same number of threads as data elements in the fractal and map blocks properly onto the embedded Sierpiński gasket*? (see Figure 4).
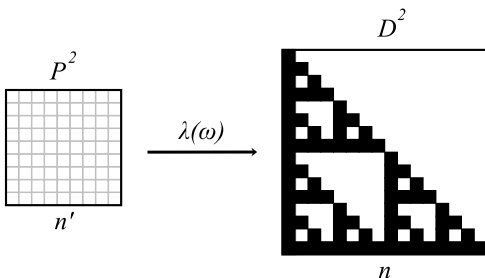

Fig. 4: A $\lambda(\omega)$ map would use asymptotically the same number of threads as data elements.

It is important to note that Question 1 asks for a block-space map and not a thread-space one. The change from thread-space to block-space allows coalesced memory to be preserved throughout the entire domain as thread organization is not compromised inside a block.

The second question relates to performance: *Will the parallel-space improvement translate into a significant GPU performance improvement*?

The present work focuses on these two questions and provides positive answers for both of them. A dedicated analysis is devoted to show that an alternating unrolling strategy allows to define a parallel-space efficient $\lambda(\omega)$ that only requires $\mathcal{O}(n^{\mathbb{H}})$ threads, with $\mathbb{H} \approx 1.58...$ being the Hausdorff dimension of the Sierpiński fractal. It addition, it is shown that by taking advantage of block-parallelism, $\lambda(\omega)$ becomes computable in $\mathcal{O}(\log_2 \log_2(n))$ time which is fast enough to produce monotonically increasing speedup once $n > n_0$ with n_0 being a constant threshold value.

The rest of the manuscript presents related work (Section II), a formal definition and analysis of $\lambda(\omega)$ (Section III) and Section (IV) presents performance results. A discussion and comments on future work is found in Section V.

II. RELATED WORK

The following related works can be classified into two categories; (1) studies on efficient GPU mapping for triangular domains and (2) general studies on the structure of the discrete Sierpiński fractal.

One of the first works that explored the possibilities of improving the GPU mapping and stage was the research of Jung *et. al.* [11] whom proposed packed data structures for representing triangular and symmetric matrices with applications to LU and Cholesky decomposition [8]. The strategy is based on building a *rectangular box strategy* for accessing and storing a triangular matrix (upper or lower). Data structures become practically half the size with respect to classical methods based on the full matrix. The strategy was originally intended to modify the data space (*i.e.,* the matrix), however one can apply the concept analogously to the parallel space.

Ries *et. al.* contributed with a parallel GPU method for the triangular matrix inversion [25]. The authors identified that the parallel space indeed can be improved by using a *recursive partition* of the grid, based on a *divide and conquer* strategy. The approach takes $O(\log_2(n))$ time by doing a balanced partition of the structure, from the orthogonal point to the diagonal.

Q. Avril *et. al.* proposed a GPU mapping function for collision detection based on the properties of the *upper-triangular map* [1]. The map is a thread-space function $u(x) \rightarrow (a, b)$, where x is the linear index of a thread t_x and the pair (a, b) is a unique two-dimensional coordinate in the upper triangular matrix. Since the map works in thread space, the map is accurate only in the range $n \in [0, 3000]$ of linear problem size.

Navarro, Hitschfeld and Bustos have proposed a block-space map function for 2-simplices and 3-simplices [18], [17], based on the solution of an m order equation that is formulated

from the linear enumeration of the discrete elements. The authors report performance improvement for 2-simplices, and for the 3-simplex case, the mapping technique is extended to the discrete orthogonal tetrahedron, where the parallel space usage can be $6\times$ more efficient. However the authors clarify that it is difficult to translate such space improvement into performance improvement, as the map requires the computation of several square and cubic roots that introduce a significant amount of overhead to the process. From the point of view of data-reorganization, a succinct blocked approach can be combined along with the block-space thread map, producing additional performance benefits with a sacrifice of $o(n^3)$ extra memory.

Exploring the benefits of efficient GPU mapping onto the embedded Sierpiński gasket becomes an interesting topic of research since its geometry is no longer Euclidean as in the related works, but instead it is embedded in an Euclidean one. Finding a proper efficient $\lambda(\omega)$ would produce an asymptotic improvement in parallel space and a potential performance improvement that could eventually be exploited.

III. ANALYSIS AND FORMULATION OF $\lambda(\omega)$

This Section first analyzes two important properties of the discrete embedded Sierpiński gasket, which are helpful in the formulation of the map $\lambda(\omega)$.

A. Analysis of the Space and Packing of $\mathcal{F}_n^{k,s}$

The discrete Sierpiński gasket belongs to a category of discrete fractals where its structure can be built by replicating k instances of itself at a scale s and placed with an arbitrary spatial organization. The notation $\mathcal{F}_n^{k,s}$ is introduced to denote such fractal, where $n \in \mathbb{N}$ is the linear size, $k \in \mathbb{N}$ the replication factor and $0 < s < 1 \in \mathbb{R}$ the scaling factor in terms of reduction. Since fractals have a recursive self-similar structure, their volume $\mathcal{V}(\mathcal{F}_n^{k,s})$ may be expressed as

$$\mathcal{V}(\mathcal{F}_n^{k,s}) = \sum_{i=1}^{k} \mathcal{V}(\mathcal{F}_{sn}^{k,s}) \qquad (1)$$

with $\mathcal{V}(\mathcal{F}_1^{k,s}) = 1$ being the limit condition of the recursion. Given that the replication factor k is fixed, and n scales by factors of s, the volume may be simplified into

$$\mathcal{V}(\mathcal{F}_n^{k,s}) = k^r \qquad (2)$$

where $r = \log_{1/s}(n)$ is defined as the scale level.

The Sierpiński gasket is a particular case where $k = 3$, $s = 1/2$ and $r = \log_2(n)$. Successive steps of the fractal produce the pattern early depicted in Figure 2. The following two lemmas are introduced to support the formulation of a map $\lambda(\omega)$.

Lemma 1: The space occupied by a discrete embedded Sierpiński gasket is in correspondence with the Hausdorff dimension of the infinite Sierpiński gasket.

Proof: The space occupied by a Sierpiński gasket of linear size n is $\mathcal{V}(\mathcal{F}_n^{3,\frac{1}{2}}) = 3^r$. Given that $r = \log_2(n)$ and $3^{log_2(x)} = 2^{\log_2(3)\log_2(x)}$, the space expression can be rearranged into

$$\mathcal{V}(\mathcal{F}_n^{3,\frac{1}{2}}) = n^{\mathcal{H}=\log_2(3)} \qquad (3)$$

where the exponent $\mathcal{H} = \log_2(3) \approx 1.5849...$ is the Hausdorff dimension of the original infinite Sierpiński gasket. ∎

Lemma 2: A discrete Sierpiński gasket of level r packs into a 2-orthotope Π^2 of dimensions $3^{\lceil \frac{r}{2} \rceil} \times 3^{\lfloor \frac{r}{2} \rfloor}$.

Proof: Proof by induction on r:

- Base case: At scale $r = 0$ the fractal has a space of $\mathcal{V}(\mathcal{F}_1^{3,\frac{1}{2}}) = 1$ element that packs into a regular 2-orthotope of $1 \times 1 = 3^{\lceil \frac{0}{2} \rceil} \times 3^{\lfloor \frac{0}{2} \rfloor}$ satisfying $3^{\lceil \frac{r}{2} \rceil} \times 3^{\lfloor \frac{r}{2} \rfloor}$.

- Induction step: It is assumed that the orthotope for $r = k$ is quasi-regular. If k is even, the packing for $k + 1$ will triple the horizontal dimension of the 2-orthotope. If k is odd, the packing for $k+1$ will triple the vertical dimension of the 2-orthotope. Since even and odd must alternate, the dimensions of the packed 2-orthotope for $k + 1$ can only be $3 \cdot 3^{\lceil \frac{k}{2} \rceil} \times 3^{\lfloor \frac{k}{2} \rfloor}$ or $3 \cdot 3^{\lceil \frac{k}{2} \rceil} \times 3 \cdot 3^{\lfloor \frac{k}{2} \rfloor}$, which is regular or quasi-regular, respectively. ∎

The packing process of Lemma (2) is illustrated in Figure 5, where the packing steps are in correspondence with the scale levels shown in Figure 5.
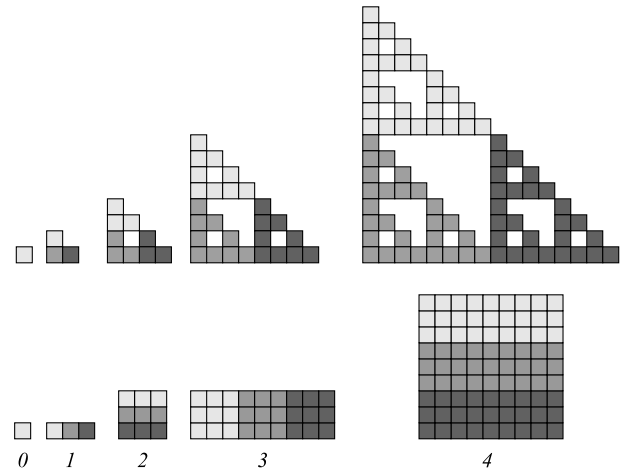

Fig. 5: Each scale of the Sierpiński fractal packs into a 2-orthotope Π^2 of dimensions $3^{\lceil \frac{r}{2} \rceil} \times 3^{\lfloor \frac{r}{2} \rfloor}$.

B. Changing from Thread-space to Block-space

An important aspect to consider is at which level the parallel space will be mapped. Two approaches are possible; (1) thread-space mapping and (2) block-space mapping. For first one, $\lambda(\omega)$ defines ω as a unique thread location in parallel space. For the second approach, $\lambda(\omega)$ defines ω as a block coordinate in which several threads are contained. The block-space approach has three important advantages over thread-space mapping. First, in block-space, the fractal becomes a simplified version of the original, requiring less elements to be mapped. Second, since the fractal is a simplified version of itself, it is possible to work on higher sizes of n before the CUDA grid maximum dimensions are reached. Third, the

block-space approach allows the possibility for threads inside a block to preserve locality, which is essential for coalesced memory accesses on data.

The block-space map $\lambda(\omega)$ is now introduced, where $\omega = (\omega_x, \omega_y)$ denotes a two-dimensional coordinate of a block of constant size $|B| = \rho_x \times \rho_y$ threads. The change from thread-space to block-space means that blocks are mapped to a simplified version of the fractal of linear size $n_b = n/b$, as shown in Figure 6.
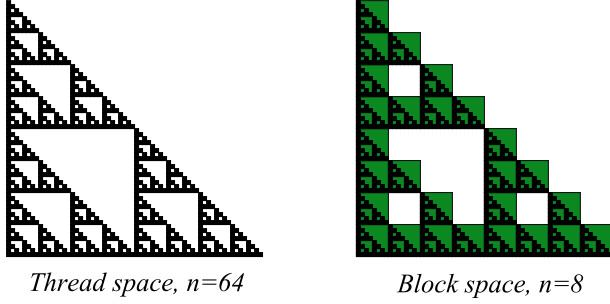

Thread space, n=64   *Block space, n=8*

Fig. 6: In thread-space mapping, threads are directly mapped one-to-one to the elements of the fractal of linear size $n = 64$. In block-space mapping, $|B| = 8 \times 8$ and blocks of threads are mapped onto a simplified version of the fractal of linear size $n_b = 64/8 = 8$.

It is important to clarify that the extra green regions visible in Figure 6 do not necessarily mean unused threads. The stage of local thread mapping is detailed later in this Section, where three approaches can be used.

The formulation of $\lambda(\omega)$ continues in block-space with $\rho_x = \rho_y$ to simplify the analysis, with n_b the new simplified linear size of the fractal with the origin $(0,0)$ located at the top-left corner for both the parallel and embedded fractal spaces, and with y increasing downwards.

C. Formulation of $\lambda(\omega)$

The function $\lambda : \mathbb{Z}_{\mathbb{E}}^2 \mapsto \mathbb{Z}_{\mathbb{F}}^2$ is introduced as a mapping of block coordinates ω in parallel-space onto block coordinates in the embedded space. The intuition behind is an unrolling process applied in parallel to each $\omega \in \Pi^2$ through all the scale levels. At each level, different x, y offsets are accumulated to form the final $(\lambda_x(\omega), \lambda_y(\omega))$ coordinate in the fractal.

Theorem 1: There exists a block-space parallel-space efficient $\lambda(\omega)$ that can map blocks in $\mathcal{O}(\log_2 \log_2(n_b))$ time using $|B| = \theta(\frac{\log_2(n_b)}{\log_2 \log_2(n_b)})$ threads per block.

Proof: By construction: let $r_b = 2^{n_b}$ be the block-space scale level of the fractal, Π^2 the 2-orthotope of $3^{\lceil \frac{r_b}{2} \rceil} \times 3^{\lfloor \frac{r_b}{2} \rfloor}$ blocks that maps onto the discrete Sierpiński gasket $\mathcal{F}_{n_b}^{3,\frac{1}{2}}$, with each block having $\rho_x \times \rho_y$ threads. By Lemma (1), Π^2 is parallel-space efficient in block-space.

A helper index function $\beta_\mu(\omega)$ is defined as

$$\beta_\mu(\omega) = \left(\frac{\omega_x((\mu+1) \mod 2) + \omega_y(\mu \mod 2)}{3^{\lceil \frac{\mu}{2} \rceil - 1}} \right) \mod 3 \tag{4}$$

to produce an index in the range $\beta_\mu(\omega) = 0, 1, 2$ that identifies, within scale level $\mu \in [0..r_b]$, which of the three regions of

the fractal does block ω belongs to. For even μ, $\beta_\mu(\omega)$ acts on ω_x. For odd μ, it acts on ω_y. Regions are sorted as 0 for top, 1 for middle and 2 for right (see Figure 1 for visual reference).

Having the $\beta_\mu(\omega)$ index, the weight functions

$$\Delta_\mu^x = \left\lfloor \frac{\beta_\mu}{2} \right\rfloor, \quad \Delta_\mu^y = \beta_\mu - \left\lfloor \frac{\beta_\mu}{2} \right\rfloor \tag{5}$$

compute the offset weights, 0 or 1, for each of the x and y directions at scale level μ. The value of the offset corresponds to $2^{\mu-1}$, which is the linear size of each region at scale level μ. For a given μ, the combination of the weight functions with the offset produce partial coordinates of the form

$$\tau_x^\mu = \Delta_\mu^x 2^{\mu-1}, \tag{6}$$
$$\tau_y^\mu = \Delta_\mu^y 2^{\mu-1} \tag{7}$$

that contribute to the final mapped coordinate. The summation of all partial coordinates produce the map

$$\lambda(\omega) = (\lambda_x(\omega), \lambda_y(\omega)), \tag{8}$$
$$\lambda_x(\omega) = \sum_{\mu=1}^{\log_2(n_b)} \tau_\mu^x \tag{9}$$
$$\lambda_y(\omega) = \sum_{\mu=1}^{\log_2(n_b)} \tau_\mu^y \tag{10}$$

which can be computed in $\mathcal{O}(\log_2 \log_2(n))$ time (*i.e.,* $n_b \in \theta(n)$) using a parallel reduction with the threads contained in the ω block. Finally, by Brent's Theorem [3], $|B| = \theta(\frac{\log_2(n)}{\log_2 \log_2(n)})$ threads are sufficient for a block of threads to reduce efficiently in parallel. ∎

Theorem 2: $\lambda(\omega)$ requires asymptotically less work than the bounding-box approach.

Proof: The asymptotic work improvement factor of $\lambda(\omega)$ with respect to the bounding-box approach is the quotient of the costs of mapping all blocks using their corresponding Π^2 structures with the consideration $n_b \in \theta(n)$

$$S_{\lambda(\omega)} = \frac{\mathcal{O}(1)\mathcal{V}(\Pi_{BB}^2)}{\mathcal{O}(\log_2 \log_2(n))\mathcal{V}(\Pi_{\lambda(\omega)}^2)} \tag{11}$$
$$\tag{12}$$

where Π_{BB}^2 and $\Pi_{\lambda(\omega)}^2$ are the parallel-spaces for the bounding-box and $\lambda(\omega)$ approaches, respectively. The parallel-space of Π_{BB}^2 corresponds to the Euclidean box of $n_b \times n_b$ blocks, and the parallel-space of $\Pi_{\lambda(\omega)}^2$ is $\mathcal{O}(n^\mathcal{H})$ by Lemma (1). Applying the limit $n \to \infty$

$$\lim_{n \to \infty} S_{\lambda(\omega)} = \lim_{n \to \infty} \frac{\frac{\partial}{\partial n}(n^{2-\mathcal{H}})}{\frac{\partial}{\partial n}(\log_2 \log_2(n))} \tag{13}$$
$$= \lim_{n \to \infty} \frac{(2-\mathcal{H})n^{1-\mathcal{H}}}{\frac{1}{n \log_2(n))}} \tag{14}$$
$$= \infty \tag{15}$$

∎

The importance of Theorem (2) is that it guarantees the existence of a $n > n_0$ where $\lambda(\omega)$ will start becoming each time faster than the bounding box approach. A theoretical
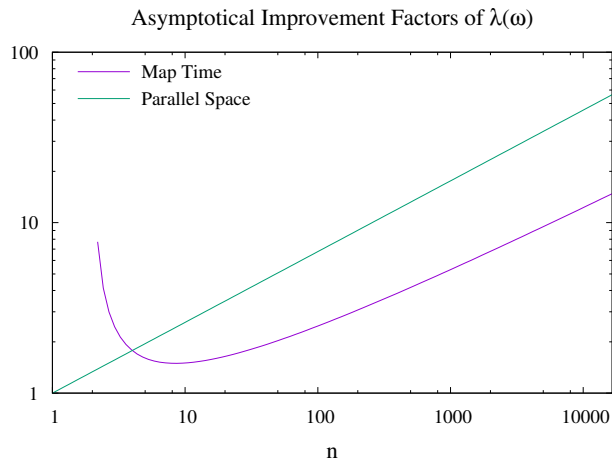
Fig. 7: The theoretical improvement for parallel-space and mapping time.

curve is presented in Figure 7. From the plot, one can note that space improvement is clear in the log-log scale. For the time improvement, the improvement decreases until $n_0 \sim 7$, which is the point where it becomes a monotonically increasing function. In practice, constants could have an effect that would push n_0 further to the right.

D. Intra-Block Mapping

Once $\lambda(\omega)$ maps a block ω, all the $\rho_x \times \rho_y$ threads contained have a shared reference coordinate that is available to use for computing their individual location in the fractal. This phase of organizing the threads within a block is referred here as *Intra-Block Mapping*, and this subsection describes three possible approaches to accomplish this.

1) Further Unrolling: In this approach threads inside their mapped block use the same $\lambda(\omega)$ map but applied to each thread. By Theorem (1), the Intra-block map is parallel-space efficient and the mapping time becomes $\mathcal{O}(\log_2 \log_2(|B|)) \in \mathcal{O}(1)$ as ρ_x, ρ_y are constant and do not grow with n.

2) Shared Lookup Table: The second approach is to use a shared lookup table of $\rho_x \times \rho_y = \mathcal{O}(1)$ offset coordinates, available to any thread in any block. Mapping each thread would cost $\mathcal{O}(1)$ memory accesses and the extra memory introduced by the shared table is $\mathcal{O}(\rho_x \times \rho_y) \in \mathcal{O}(1)$.

3) Bounding Sub-boxes: The third approach consists of using small bounding-boxes in each block. This approach introduces a constant number of extra threads in each block, but allows each thread to be mapped just with $f(x) = x$ which costs $\mathcal{O}(1)$. In order to know if it is in the fractal or not, each thread evaluates if $t_x \& (n - 1 - t_y) == 0$ is true or not, respectively, with $\&$ being the bitwise AND operator.

Regardless of which Intra-block mapping approach is chosen, the final mapping time will not surpass the $\mathcal{O}(\log_2 \log_2(n))$ time, as the blocks have a constant size of threads. Nevertheless, it is worth considering that *Further Unrolling* introduces a constant cost in mapping time. The *Shared Lookup Table* approach introduces a constant cost in memory and finally the *Bounding Sub-boxes* introduce a constant in the

number of extra threads. Choosing one or another can depend on the specific application, *i.e.*, to avoid competing with the application in the use of memory bandwidth or arithmetic operations.

IV. IMPLEMENTATION AND PERFORMANCE RESULTS

A CUDA implementation of $\lambda(\omega)$ is put under test to obtain the actual speedup for different values of n. The implementation uses the *Bounding Sub-boxes* approach to arrange threads inside each block, as it is the simplest in terms of implementation time. The parallel reduction per-block is performed using the shuffle instruction of CUDA, which allows efficient register-level communication among threads within the same warp.

The performance test consists of measuring the average time taken to write a constant value on all the elements of a Sierpiński gasket of scale level r, which is embedded in a $M_{n \times n}$ matrix filled with zeros. The configuration space is explored in the ranges $r = 0..16$ and $\rho = 1, 2, 4, 8, 16, 32$ for the scale level and block-size respectively, in order to find the optimal setting that provides the best performance for both the bounding-box and $\lambda(\omega)$ approaches. The average performance measures are taken by averaging 100 sub-averages, each one being an average time of 10 consecutive synchronized kernel calls. The standard error for each mean was below 1%. The hardware for performance test is listed in Table I.

TABLE I: Hardware used for performance tests.

Device	Model
GPU	Titan-X Pascal, GP102, 3584 cores, 12GB
CPU	Intel i7-6950X 10-core Broadwell
RAM	128GB DDR4 2400MHz

Figure 8 presents the speedup of $\lambda(\omega)$ over the bounding-box approach, as well as the running times for the two mapping techniques. For values of $n < 2^8$, one can note that only some curves offer speedup. Once $n > 2^8$, the speedup begins to increase for all block-size configurations, reaching the higher values at $n = 2^{16} = 65536$, which was the highest problem size that fit in the GPU memory. An important aspect to note from the speedup curve is that for the largest possible block size, $|B| = \rho \times \rho = 32 \times 32$, the $\lambda(\omega)$ map runs the test approximately $6\times$ faster than the bounding-box approach. Furthermore, as blocks become smaller in ρ, the improvement increases dramatically, reaching up to $55\times$ of speedup.

The plot of the running times provides further insights on what configuration is the best suited for each mapping technique. By looking at the running times of the small block configurations, one can note that regardless of their high speedup, their running times are the slowest ones. For the bounding-box approach (BB) the best performance is obtained when the block-size is maximum. For $\lambda(\omega)$ the best performance is found when using a block of $|B| = 16 \times 16$ threads. If the curves of the best configuration for each implementation are considered, *i.e.*, the ones with marked points on Figure 8, right, then the speedup provided by $\lambda(\omega)$ increasing further more reaching almost an order of magnitude. The other running
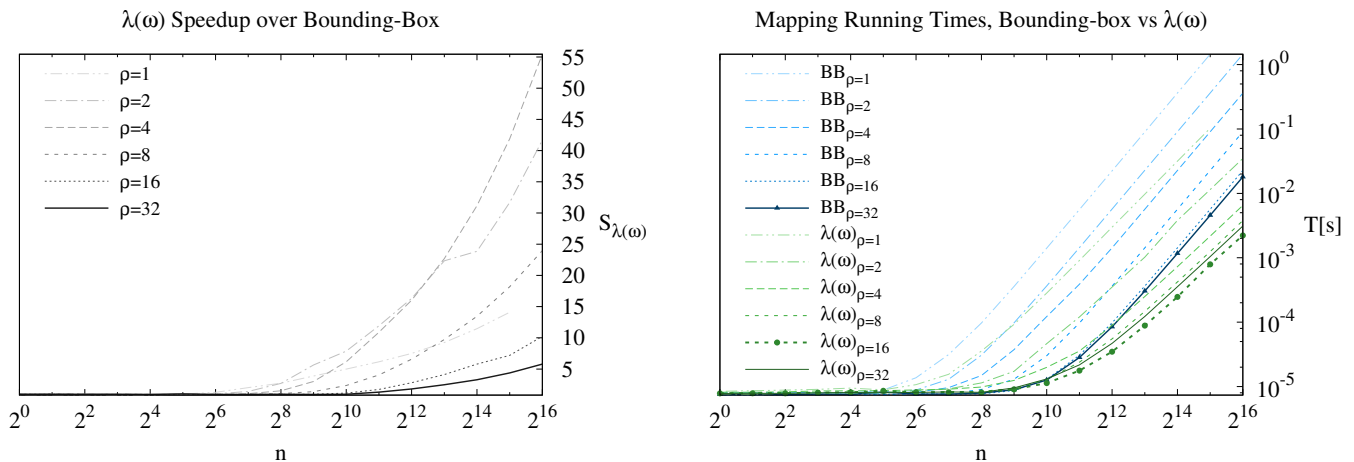
Fig. 8: On the left, the speedup of $\lambda(\omega)$ with respect to the bounding-box approach at different block-size configurations. On the right, their absolute running times at different block-size configurations.

time curves are still useful to visualize that as blocks become smaller, the value of n_0 where $\lambda(\omega)$ is more convenient moves closer to the origin, and vice versa, however the GPU with its current organization and architecture is not fully utilized by such small block configurations, thus leading to an inferior performance. Therefore, in practice large blocks should be utilized and by Theorem (2), beyond $n = 2^{16}$ the speedup would keep increasing in favor of $\lambda(\omega)$.

V. DISCUSSION

The results obtained in this work have shown that a parallel-space efficient mapping function can lead to significant performance gains in applications that require processing an embedded Sierpiński gasket. The analysis and formulation of $\lambda(\omega)$ has provided three important results; (1) There exists a correspondence between a quasi-regular 2-orthotope of discrete elements and the elements of the embedded Sierpiński gasket fractal. (2) Such correspondence from parallel to data space can be computed in $\mathcal{O}(\log_2 \log_2(n))$ time using a block-space map that is based on efficient parallel reductions. (3) The total work of mapping the 2-orthotope with $\lambda(\omega)$ is asymptotically smaller than the work generated by the bounding box approach, leading to a monotonically increasing speedup that is guaranteed to occur once a n_0 value is reached.

The experimental performance results confirm the theory as once the fractal reaches the linear size $n_0 = 2^8$, the speedup begins to increase monotonically for all block-sizes. Using the maximum block size of 32×32 threads, $\lambda(\omega)$ reached up to $6\times$ of speedup. The maximum speedup obtained was approximately $55\times$ with blocks of 1 thread, however such block configuration is not practical for the GPU architecture. Still, having measured the running times with different block configurations helped in understanding that reducing the block size only brings the n_0 value closer to the origin and increasing the block-size pushes it forward. Eventually, all block configurations can reach up to $55\times$ of speedup and beyond if the fractal is large enough.

The GPU map found for the embedded Sierpiński gasket may be adapted to work for other types of embedded fractals

that follow a similar building scheme applying modifications to the helper, weight and offset functions. In order to obtain speedup, it is crucial to check if the overall work will be asymptotically smaller than the bounding-box approach.

Two important questions may be formulated from the results obtained in this work. The first one is: *Can there exist a general $\lambda(\omega)$ that maps a family of embedded fractals who share the same building principle?*, and the second question: *can there exist a $\lambda(\omega)$ that maps in $\mathcal{O}(1)$ time using no more than $\mathcal{O}(1)$ extra memory?*. Future research in these directions can provide important insights on the limits of efficient GPU computing for embedded fractal domains.

REFERENCES

[1] Quentin Avril, Valérie Gouranton, and Bruno Arnaldi. Fast collision culling in large-scale environments using gpu mapping function. In *EGPGV*, pages 71–80, 2012.

[2] C. P. Baliarda, C. B. Borau, M. N. Rodero, and J. R. Robert. An iterative model for fractal antennas: application to the sierpinski gasket antenna. *IEEE Transactions on Antennas and Propagation*, 48(5):713–719, May 2000.

[3] Richard P. Brent. The parallel evaluation of general arithmetic expressions. *J. ACM*, 21(2):201–206, April 1974.

[4] David Doty. Theory of algorithmic self-assembly. *Commun. ACM*, 55(12):78–88, December 2012.

[5] A. Gamba, D. Ambrosi, A. Coniglio, A. de Candia, S. Di Talia, E. Giraudo, G. Serini, L. Preziosi, and F. Bussolino. Percolation, morphogenesis, and burgers dynamics in blood vessels formation. *Phys. Rev. Lett.*, 90:118101, Mar 2003.

[6] Y Gefen, A Aharony, Y Shapir, and B B Mandelbrot. Phase transitions on fractals. ii. sierpinski gaskets. *Journal of Physics A: Mathematical and General*, 17(2):435, 1984.

[7] Yuval Gefen, Benoit B. Mandelbrot, and Amnon Aharony. Critical phenomena on fractal lattices. *Phys. Rev. Lett.*, 45:855–858, Sep 1980.

[8] Fred Gustavson. New generalized data structures for matrices lead to a variety of high performance algorithms. In Roman Wyrzykowski, Jack Dongarra, Marcin Paprzycki, and Jerzy Wasniewski, editors, *Parallel Processing and Applied Mathematics*, volume 2328 of *Lecture Notes in Computer Science*, pages 418–436. Springer Berlin / Heidelberg, 2006.

[9] Kai He, Cheng-Yan Xu, Liang Zhen, and Wen-Zhu Shao. Fractal growth of single-crystal -fe2o3: From dendritic micro-pines to hexagonal micro-snowflakes. *Materials Letters*, 62(45):739 – 742, 2008.

[10] Min Chen Jian Shang, Wang Yongfeng et al. Assembling molecular Sierpiński triangle fractals. *Nat Chem*, 7(5):389–393, May 2015.

[11] Jin Hyuk Jung and Dianne P. OLeary. Exploiting structure of symmetric or triangular matrices on a gpu. Technical report, University of Maryland, 2008.

[12] James I. Lathrop, Jack H. Lutz, and Scott M. Summers. Strict self-assembly of discrete sierpinski triangles. *Theoretical Computer Science*, 410(4):384 – 405, 2009.

[13] Benoit B Mandelbrot. The fractal geometry of nature. 1982. *San Francisco, CA*, 1982.

[14] Benoit B. Mandelbrot. *Fractals*. John Wiley & Sons, Inc., 2004.

[15] Bruce T. Milne. Measuring the fractal geometry of landscapes. *Applied Mathematics and Computation*, 27(1):67 – 79, 1988.

[16] Klauko P. Mota and Paulo Murilo C. de Oliveira. Monte carlo simulations for the slow relaxation of crumpled surfaces. *Physica A: Statistical Mechanics and its Applications*, 387(24):6095 – 6104, 2008.

[17] Cristóbal A. Navarro, Benjamín Bustos, and Nancy Hitschfeld. Potential benefits of a block-space GPU approach for discrete tetrahedral domains. In *CLEI-2016, XLII Conferencia Latinoamericana de Informática, Valparaiso, Chile, October 10-14, 2016*, 2016.

[18] Cristobal A. Navarro and Nancy Hitschfeld. GPU maps for the space of computation in triangular domain problems. In *2014 IEEE International Conference on High Performance Computing and Communications, 6th IEEE International Symposium on Cyberspace Safety and Security, 11th IEEE International Conference on Embedded Software and Systems, HPCC/CSS/ICESS 2014, Paris, France, August 20-22, 2014*, pages 375–382, 2014.

[19] Cristobal A. Navarro, Nancy Hitschfeld-Kahler, and Luis Mateu. A survey on parallel computing and its applications in data-parallel problems using GPU architectures. *Commun. Comput. Phys.*, 15:285–329, 2014.

[20] Fumio Ohi and Yoshikazu Takamatsu. Time-space pattern and periodic property of elementary cellular automata — sierpinski gasket and partially sierpinski gasket —. *Japan Journal of Industrial and Applied Mathematics*, 18(1):59, 2001.

[21] Peter E. Oppenheimer. Real time design and animation of fractal plants and trees. *SIGGRAPH Comput. Graph.*, 20(4):55–64, August 1986.

[22] Michael W. Palmer. Fractal geometry: a tool for describing spatial patterns of plant communities. *Vegetatio*, 75(1):91–102, 1988.

[23] A. P. Pentland. Fractal-based description of natural scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):661–674, Nov 1984.

[24] C. Puente-Baliarda, J. Romeu, R. Pous, and A. Cardama. On the behavior of the sierpinski multiband fractal antenna. *IEEE Transactions on Antennas and Propagation*, 46(4):517–524, Apr 1998.

[25] Florian Ries, Tommaso De Marco, Matteo Zivieri, and Roberto Guerrieri. Triangular matrix inversion on graphics processing unit. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 9:1–9:10, New York, NY, USA, 2009. ACM.

[26] Winfree E Rothemund PWK, Papadakis N. Algorithmic self-assembly of dna sierpinski triangles. *PLoS Biol*, 2(12):e424, 2004.

[27] E. R. Weibel. Fractal geometry: a design principle for living organisms. *American Journal of Physiology - Lung Cellular and Molecular Physiology*, 261(6):L361–L369, 1991.

[28] Stephen Wolfram. Statistical mechanics of cellular automata. *Rev. Mod. Phys.*, 55(3):601–644, July 1983.