Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 587–593
ISBN 978-83-60810-51-4

A parallel searching algorithm for the insetting procedure in Matlab Parallel Toolbox

Dimitris N. Varsamis*, Paris A. Mastorocostas*, Apostolos K. Papakonstantinou† and Nicholas P. Karampetakis‡
*Department of Informatics & Communications
Technological Educational Institute of Serres, Serres 62124, Greece
Email: dvarsam@teiser.gr, mast@teiser.gr
†Cartography and Geoinformation Lab, Department of Geography
University of the Aegean, Mytilene 81100, Greece
Email: apapak@geo.aegean.gr
‡Department of Mathematics
Aristotle University of Thessaloniki, Thessaloniki 54124, Greece
Email: karampet@math.auth.gr

***Abstract*—In this paper we present the implementation of a parallel searching algorithm, which is used for the insetting procedure in cartography. The calculation time of the above procedure is very long due to the fact that the datasets in cartography are maps with large and very large resolution. The purpose of this proposal is to reduce the calculation time in a multicore machine with shared memory. The proposed algorithm and the performance tests are developed in Matlab Parallel Toolbox.**

I. Introduction

CARTOGRAPHY often deals with problems that confront the necessity of designing cartographic software tools. One of these problems faced by cartographers is the insetting procedure. In this procedure mapmakers are placing insets into the maps, in order to tackle with the land discontinuity problem [1], [2].

Papakonstantinou et al. in [2], [3] presented a software tool called Inset Mapper (IM), that helps cartographers to overcome insetting procedure difficulties. In this approach, a computational procedure following specific cartographic rules for calculating the appropriate position and the optimal scale for an inset was presented [4], [5]. In this computational procedure, a modified searching algorithm is used in order to calculate the number and the position of the proposed insets. This algorithm runs repetitively to calculate inset dimensions until the computational process finds the optimal results. Both the computational procedure and the searching algorithm were developed in Matlab and were embedded into the IM software tool. The calculation time of this computational procedure and more specific the searching algorithm execution time depends on the dataset size and is defined from a large degree polynomial function. In very large geographic datasets the execution time problem becomes more significant,therefore it is essential to study efficient ways of parallel processing in order to reduce it.

This work was supported by the Research Committee of the Serres Institute of Education and Technology

Parallel processing is one of the most successful efforts to introduce explicit parallelism to high level programming languages [6], [7], [8]. This approach is selected because many useful computations can be conducted in terms of a set of independent sub-computations, each strongly associated to an element of a large dataset, where such computations are inherently parallelizable [9], [10]. Parallel processing is particularly convenient because it can scale easily to large problem sizes, such as searching into large geographic datasets.

The aim of this paper is to present a parallel processing approach in order to reduce the calculation time of the computational procedure and the total execution time of the IM software tool, respectively. In particular, in section 2 we present the serial searching algorithm. The theoretical number of iterations and the theoretical computational time are computed as well. In section 3 the modification of the above algorithm that is implemented using parallel processing is presented. Additionally, in this section we compute the corresponding theoretical number of iterations, while the theoretical calculation times for the parallel processing implementations is also given. Finally, in section 4 we apply the implemented algorithms using the parallel computing toolbox of Matlab [11], [12], [13] and a comparison of execution times for specific geographic datasets is conducted.

II. The Serial Searching Algorithm

In this section we present the modified for the insetting procedure serial searching algorithm, witch is based on the sequential searching algorithm. In particular, the modified algorithm is implemented using two double loops in order to find the total number of inset placement positions. Additionally, we compute the number (L) of iterations and the bounded interval of L. Finally, the computational complexity of this algorithm is given.

Algorithm 1 Serial processing

Input: A, m_i, n_i
Output: $N, positions$
for $i = 1$ to $m - m_i + 1$ **do**
  $j \leftarrow 1$
  while $j \leq n - n_i + 1$ **do**
    $B \leftarrow A(i$ to $i + m_i - 1, j$ to $j + n_i - 1)$
    $bj \leftarrow n_i$
    $done \leftarrow true$
    while $bj \geq 1$ **do**
      $bi \leftarrow 1$
      while $bi \leq m_i$ **do**
        if $b(bi, bj) = 1$ **then**
          $done \leftarrow false$
          $pos \leftarrow bj + 1$
          Break
        end if
        $bi \leftarrow bi + 1$
      end while
      $bj \leftarrow bj - 1$
    end while
    if $done = true$ **then**
      Count and Store position
      $j \leftarrow j + 1$
    else
      $j \leftarrow pos$
    end if
  end while
end for

The primary aim of this algorithm is to compute the number N of the proposed inset map placement positions in a map using predefined inset dimensions. The geographic dataset (map) is transformed to a two-dimensional $(m \times n)$ numerical matrix, where n is the length and m the width of the dataset in pixels. This matrix has values, for every element zero corresponding to sea and one to land:

$$a_{i,j} = \begin{cases} 1, & land \\ 0, & sea \end{cases} \tag{1}$$

Let a matrix $A \in \mathbb{K}^{m \times n}$ and the matrix $B \in \mathbb{K}^{m_i \times n_i}$, being sub-matrix of A, where $\mathbb{K} = \{0, 1\}$. The dimensions of matrix B are $m_i \times n_i$, where m_i and n_i are the length and the width (in pixels) of the proposed inset, respectively.
In mathematical representation, the problem is to count (and store positions) how many times the matrix B exists into the matrix A having all the elements equal to zero.
The serial searching algorithm is described in Algorithm 1.

In this algorithm the number of iterations depends on the size of matrix A and B. Additionally, for both internal and external loops the number of iterations depends on the elements of matrix B, namely on the existence of the value one. The external double loop is defined by the indices i and j and the internal double loop is defined by the indices bj and bi, respectively. In the external double loop the number of iterations for index i is constant and equal to $m - m_i + 1$, while the number of iterations for index j is between 1 and $n-n_i+1$. Thus, the maximum amount of iterations for external loop is $(n - n_i + 1) \times (m - m_i + 1)$.
In the internal loop the respective amount is $n_i \times m_i$ iterations. Therefore, the maximum total number of iterations that are executed in the serial algorithm, where the inputs are $A^{m \times n}$, m_i and n_i, is given by

$$L_{max}(n, m) = (n - n_i + 1) \cdot (m - m_i + 1) \cdot n_i \cdot m_i \tag{2}$$

Consequently, the theoretical complexity of the above algorithms is of $O\left(n^2 m^2\right)$.
Specifically, in the external loop, the number of iterations for each row (index j) depends on the coverage of the map, where for matrix A the coverage is given by

$$C = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{i,j}}{m \cdot n}$$

where $a_{i,j} \in A^{m \times n}$.
Therefore, the number of iterations for each row is $(n - n_i + 1) \times (1 - C)$.
Similarly, in the internal loop the number of iterations depends on the coverage of map, thus it is equal to $(m - m_i + 1) \times (1 - C)$.
We conclude that the total number of iterations is

$$L = (n - n_i + 1)\,(m - m_i + 1)\,n_i m_i \times (1 - C)^2$$

According to the following cartographic rules from [3], [14] we have that

$$\frac{n \cdot m}{16} \leq n_i \cdot m_i \leq \frac{n \cdot m}{8}$$

In case where $n \simeq m$ and $n_i \simeq m_i$, he have

$$\frac{n^2}{16} \leq n_i^2 \leq \frac{n^2}{8} \text{ and } \frac{m^2}{16} \leq m_i^2 \leq \frac{m^2}{8}$$

or equivalently

$$\frac{n}{4} \leq n_i \leq \frac{n}{\sqrt{8}} \text{ and } \frac{m}{4} \leq m_i \leq \frac{m}{\sqrt{8}} \tag{3}$$

Therefore, the total number of iterations is enclosed

$$L_1 \leq L \leq L_2$$

where

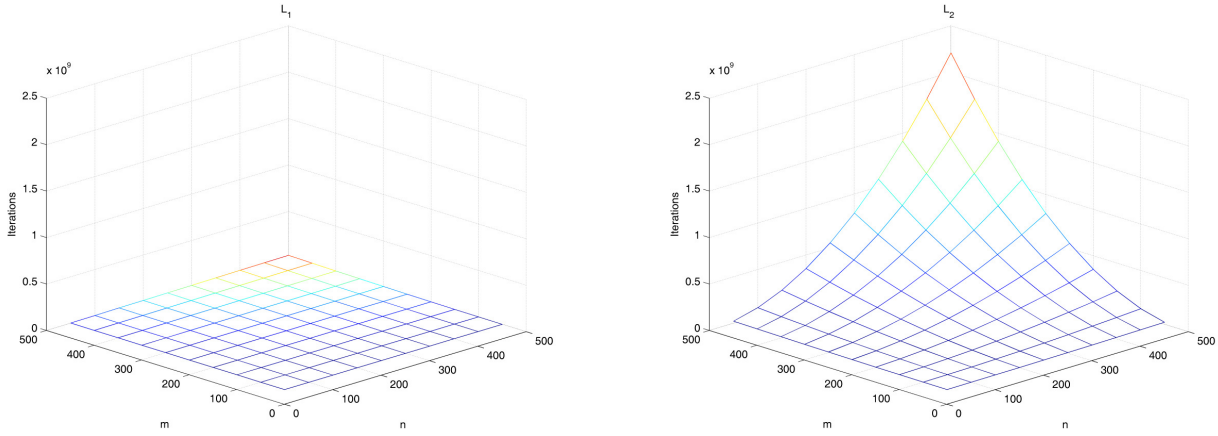
$$L_1 = \frac{\left((\sqrt{8}-1)n + \sqrt{8}\right)\left((\sqrt{8}-1)m + \sqrt{8}\right)nm}{8^2} \times (1-C)^2$$

and

$$L_2 = \frac{(3n+4)(3m+4)nm}{4^4} \times (1-C)^2$$

The number of iterations is increased rapidly while the dimensions of map are increased (Figure 1a and 1b).
In case where $m = 200$ and $n = 200$, we have

$$0.19148 \times 10^7 \leq L \leq 5.70025 \times 10^7$$



(a) L_1 (b) L_2

Fig. 1. The lower and upper bound of iterations

while in case where $m = 500$ and $n = 500$, we have

$$2.9781 \times 10^7 \leq L \leq 220.9 \times 10^7$$

The increase in L has as a result a significant influence on the execution time of the serial algorithm and the total calculation time of the computational procedure of the IM software tool.

III. The Parallel Searching Algorithm

The serial algorithm access all rows of the matrix A, and for each row checks if there exists an inset map placement position; this procedure is independent for each row. Also, the number of iterations for index i (rows) in the external loop in matrix A is fixed. Thus, we can partition the loop for index i (rows in matrix A) into a $(m - m_i + 1)$ independent tasks (Figure 2). These tasks can run simultaneously in a set of processors. Each of these tasks performs the same procedure for a different dataset. The architecture of parallel processing for this implementation is a Single Instruction Multi Data (Figure 3) with shared memory. From the above we conclude that the most simple way to parallelize the serial algorithm [10], [9] is to parallelize the external loop for index i, because this loop has a constant number of iterations since this procedure can be applied to each row separately.
The parallel searching algorithm is described in Algorithm 2.
The number of iterations in each processor (row) is given by

$$L_p = (n - n_i + 1) \cdot n_i \cdot m_i \cdot (1 - C)^2$$

The computation time of the serial algorithm is given by

$$T_1 = L \cdot t_c = L_p \cdot (m - m_i + 1) \cdot t_c$$

where t_c is the computational time of each iteration.
The computation time of parallel algorithm is given by

$$T_p = (L_p \cdot t_c + p \cdot T_{comm}) \cdot \frac{(m - m_i + 1)}{p}$$

Algorithm 2 Parallel processing

Input: A, m_i, n_i
Output: $N, positions$
for $i = 1$ to $m - m_i + 1$ in Parallel **do**
  $j \leftarrow 1$
  while $j \leq n - n_i + 1$ **do**
    $B \leftarrow A(i$ to $i + m_i - 1, j$ to $j + n_i - 1)$
    $bj \leftarrow n_i$
    $done \leftarrow true$
    while $bj \geq 1$ **do**
      $bi \leftarrow 1$
      while $bi \leq m_i$ **do**
        if $b(bi, bj) = 1$ **then**
          $done \leftarrow false$
          $pos \leftarrow bj + 1$
          Break
        end if
        $bi \leftarrow bi + 1$
      end while
      $bj \leftarrow bj - 1$
    end while
    if $done = true$ **then**
      Count and Store position
      $j \leftarrow j + 1$
    else
      $j \leftarrow pos$
    end if
  end while
end for

where p is the number of processors and T_{comm} is the required time for communication between CPUs and memory.
Therefore, the theoretical speed up of the parallel algorithm is

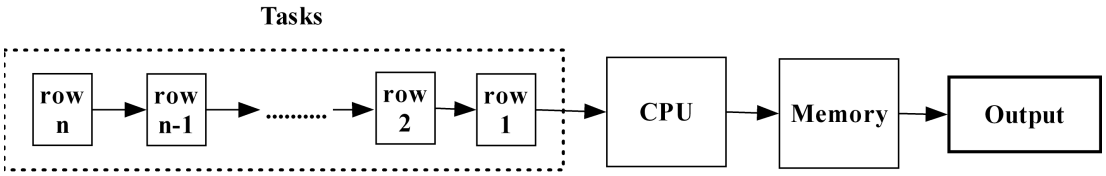


Fig. 2. Serial processing architecture

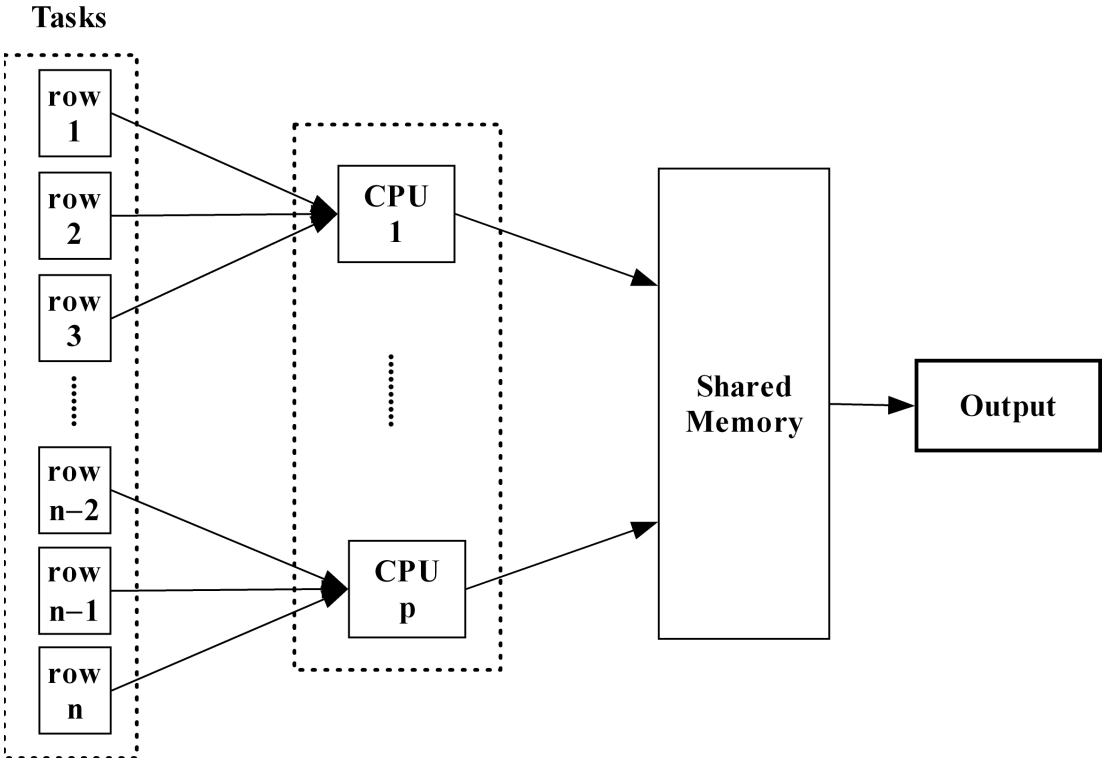


Fig. 3. Parallel processing architecture

given by

$$S_p = \frac{T_1}{T_p} = \frac{L_p \cdot (m - m_i + 1) \cdot t_c}{(L_p \cdot t_c + p \cdot T_{comm}) \cdot \frac{(m-m_i+1)}{p}} = \frac{L_p \cdot t_c \cdot p}{L_p \cdot t_c + p \cdot T_{comm}}$$

and the efficiency of this algorithm is given by

$$E_p = \frac{S_p}{p} = \frac{L_p \cdot t_c}{L_p \cdot t_c + p \cdot T_{comm}}$$

In case where $T_{comm} = a \cdot t_c$ we have

$$S_p = \frac{L_p \cdot t_c \cdot p}{L_p \cdot t_c + a \cdot t_c} = \frac{L_p \cdot p}{L_p + p \cdot a}$$

and

$$E_p = \frac{S_p}{p} = \frac{L_p}{L_p + p \cdot a}$$

A common practice for the insetting procedure in cartography is to create insets with the dimensions' proportions similar to the proportions of the selected area, in order to have a balanced design into the final result [14]. According to the previous condition, for the realisation of the case study, we selected the inset dimensions to follow the ratio of the map dimensions for arbitrary area to inset. We have to point out that the efficiency E_p of the parallel algorithm depends on L_p, p and a. In particular, if the inset dimensions are $m_i = \frac{m}{4}$ and $n_i = \frac{n}{4}$ and a map has land coverage $C = 0.40$, then

$$L_p = \left(n - \frac{n}{4} + 1\right) \cdot \frac{n}{4} \cdot \frac{m}{4} \cdot (1 - C)^2 = \frac{9}{1600}\left(3n^2 + 4n\right) m$$

For a map with resolution 100 dpi the dimensions are $m = 400$ and $n = 400$, thus the number L_p is very large and the efficiency converge to one.

IV. Implementation in Matlab Parallel Toolbox

The performance tests for both serial and parallel algorithms are implemented in Matlab Parallel Toolbox. In this toolbox the loop command "parfor" is included, which implements the parallelization of simple loop command "for". The parallel algorithm is constructed in Matlab programming language

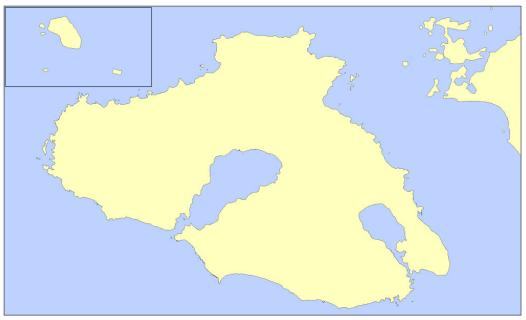


Fig. 4. The island of Lesvos with inset

with the appropriate features. The following performance tests are implemented in a personal computer using the software Matlab, having the specifications: Intel Core Quad CPU (Q9400) at 2600 GHz with 4 Gb RAM.

In the performance tests we measure the execution time of the serial algorithm (1 core) and the parallel algorithm with two cores and four cores.

The geographical dataset is the map of island Lesvos (Figure 4) with resolution varying from 50 to 500 dpi, with coverage $C = 0.40$. According to (3) we study the following four cases for the inset dimensions:

1) $m_i = \frac{m}{4}$ and $n_i = \frac{n}{4}$
2) $m_i = \frac{2m}{7}$ and $n_i = \frac{2n}{7}$
3) $m_i = \frac{3m}{10}$ and $n_i = \frac{3n}{10}$
4) $m_i = \frac{m}{3}$ and $n_i = \frac{n}{3}$

In case (4) the number N of inset placement position is equal to zero which means that the study of this case does not have cartographic interest. In the other three cases, the number of inset placement positions is unequal to zero and has the order $N_3 < N_2 < N_1$. Therefore, the forthcoming analysis is takes into consideration cases (1), (2) and (3).

In Figures 5, 6 and 7 we present the execution time (in seconds) of the serial (1 core) and parallel (2 cores, 4 cores) algorithms for each case, respectively. An increase in the map resolution leads to increase in corresponding time. The inset dimensions have an effect on the execution time because in case (1) the serial algorithm (1 core) and the parallel algorithm (2 cores) have the same time, but in cases (2) and (3) the divergence between one core, two cores and four cores increases.

The results of performance tests are hosted in Table I where the speed up of the parallel algorithm is presented. Figure 8 depicts the efficiency of parallel algorithm.

The speed up of the parallel algorithm for implementation with two and four cores is greater than to one ($S_p \geq 1$) in all cases, except case (3) with two cores. Generally, we have improvement in the absolute execution time of the parallel algorithm instead of the serial one.

The speed up for implementation with four cores is greater than that of two cores in all cases. Thus, in a machine with

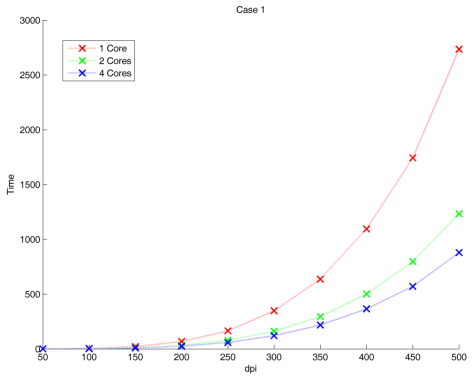


Fig. 5. Execution times for serial (1 core) and parallel (2 and 4 cores) algorithms for the case (1)

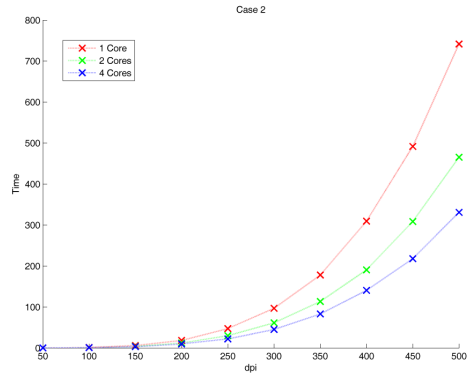


Fig. 6. Execution times for serial (1 core) and parallel (2 and 4 cores) algorithms for the case (2)

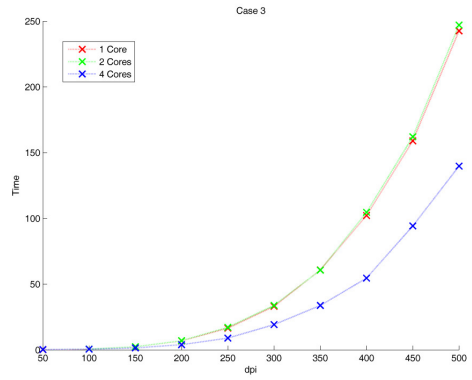
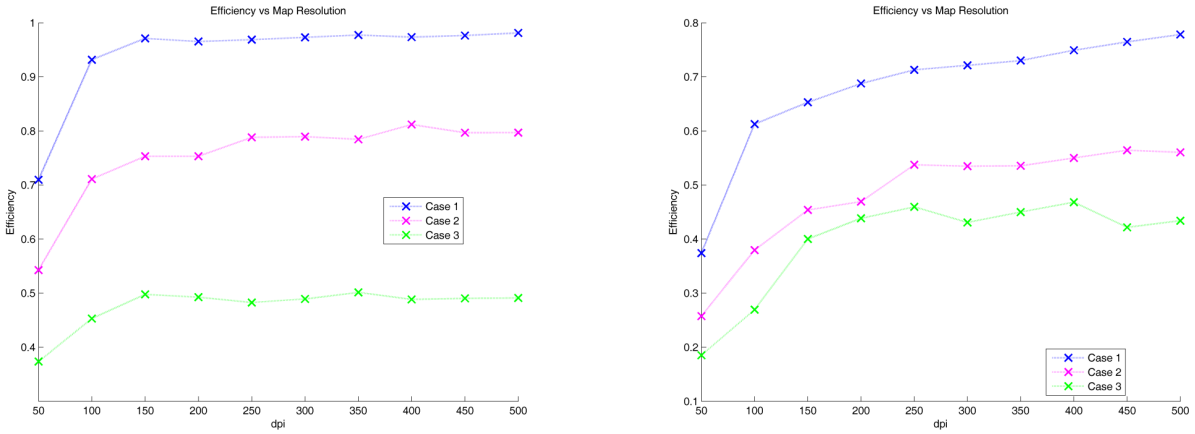


Fig. 7. Execution times for serial (1 core) and parallel (2 and 4 cores) algorithms for the case (3)

TABLE I
THE PARALLEL ALGORITHM SPEED UP FOR ALL IMPLEMENTATIONS

	Case 1		Case 2		Case 3	
	S_2	S_4	S_2	S_4	S_2	S_4
50 dpi	1.419180	1.496161	1.084891	1.029603	0.746859	0.740214
100 dpi	1.863052	2.450053	1.421818	1.518030	0.905782	1.077569
150 dpi	1.942125	2.611736	1.506132	1.815267	0.995336	1.600386
200 dpi	1.930686	2.750511	1.506229	1.876751	0.985022	1.753372
250 dpi	1.937525	2.851550	1.576129	2.149487	0.965037	1.837854
300 dpi	1.946198	2.885138	1.578884	2.138818	0.978388	1.722617
350 dpi	1.954530	2.920274	1.568819	2.141944	1.002680	1.799576
400 dpi	1.946883	2.996352	1.623641	2.199886	0.976638	1.872199
450 dpi	1.952799	3.058527	1.593354	2.257043	0.980677	1.686572
500 dpi	1.962332	3.113050	1.593543	2.241439	0.981906	1.735417



(a) 2 Cores (b) 4 Cores

Fig. 8. The parallel algorithm Efficiency for all implementations

quad core processor we prefer the four core implementation.

The maximum efficiency for implementation with two cores is achieved in all cases in the map with the resolution of 150 dpi. Then the efficiency is stable and equal to 95% (excellent) in case (1), equal to 75% (good) in case (2) and equal to 50% (poor) in case (3).
The efficiency for implementation with four cores is upward but in all cases is less than the corresponding one of the implementation with two cores. Thus this implementation can be applied in maps with resolution greater than 500 dpi.
The efficiency depends on the number of inset placement positions, because the order of efficiency in cases (1), (2) and (3) follow the order of numbers $N_3 < N_2 < N_1$.

From the above, we conclude that the execution time of searching algorithm is reduced by the use of parallel processing. In case where the cartographers select random inset dimensions following the cartographic rules [3], [14] the implementation with four cores gives better results in all cases. It should be noted that in SIMD parallel architecture in Matlab Parallel Toolbox with shared memory, the size of memory has significant role. The time and the efficiency of the parallel algorithm is improved in the presence of a lot of memory. We have to point out that in maps with resolution higher than 700 dpi (for the computer specs mentioned above) the Matlab confronts memory problems. This can be limited by using distributed memory in a network of computers or a grid computer.

V. CONCLUSION

A parallel algorithm for the insetting procedure in a multicore machine with shared memory has been given. The total number of iterations for both the serial and the parallel algorithms has been computed. The theoretical computational time of the parallel algorithm is better than the serial one and is depended on the number of processors. The efficiency of the parallel algorithm is in general very good, helping the cartographers who use the IM software tool to reduce its execution time in a local machine with multicore processor.

In the future, we aim to study the speed-up and the efficiency of the parallel algorithm, and to transform the IM software tool to a web-based application in parallel architecture with distributed memory in a network of computers or in a grid computer.

REFERENCES

[1] F. Ormeling, "Island cartography," in *Proceedings of the 7th National Cartographic Conference*, Lesvos, Greece, 2002, pp. 22–30.

[2] A. Papakonstantinou, D. Varsamis, and N. Soulakellis, "Inset mapper: A software tool in island cartography," *Cartography and Geographic Information Science*, vol. 38, no. 4, pp. 384–397, 2011.

[3] A. Papakonstantinou, D. Varsamis, N. Soulakellis, and P. Kyriakidis, "Inset mapper: A tool for scale selection and inset placing configuration of multiple insets in island cartography," in *Proceedings of the 25th International Cartographic Conference*, Paris, France, 2011, co-308.

[4] A. Robinson, J. L. Morrison, P. C. Muehrcke, J. A. Kimerling, and S. C. Guptill, *Elements of Cartography*, 5th ed. New York, USA: John Wiley & Sons Inc, 1995.

[5] J. A. Tyner, *Principles of Map Design*, 1st ed. New York, USA: The Guilford Press, 2010.

[6] B. Wilkinson and M. Allen, *Parallel Programming*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2004.

[7] C. Lin and L. Snyder, *Principles of Parallel Programming*. Boston, USA: Addison-Wesley, 2008.

[8] T. Rauber and G. Runger, *Parallel Programming for Multicore and Cluster Systems*. New York, USA: Springer, 2010.

[9] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

[10] J. W. Demmel, M. T. Heath, and H. A. van der Vorst, "Parallel numerical linear algebra," *Acta Numerica*, vol. 2, pp. 111–197, 1993.

[11] J. Kepner, *Parallel MATLAB for Multicore and Multinode Computers*. Philadelphia, USA: SIAM, 2009.

[12] P. Luszczek, "Parallel programming in matlab," *International Journal of High Performance Computing Applications*, vol. 23, no. 3, pp. 277–283, 2009.

[13] G. Sharma and J. Martin, "Matlab : A language for parallel computing," *International Journal of Parallel Programming*, vol. 37, pp. 3–36, 2009.

[14] G. Peterson, *GIS cartography: a guide to effective map design*. CRC Press, 2009.