In this work we are concerned with Blake–Zissermann [1] (BZ) variational method for image segmentation: this approach leads to a minimization of a non-convex objective energy functional. Very often, the segmentation of large-size gridded data is addressed via tiling procedure: additional specific post-processing on tile boundaries may be needed in order to reduce the effect of the subdivision. We aim to show that a simple tiling strategy enables us to treat large images even in a commodity multicore CPU, with no need of specific post-processing on tile junctions.

Blake–Zissermann continuous model

Continuous model can be stated as:

\[ F(x, z, u) = \frac{\beta}{2} \| \nabla u \|^2 + \frac{\alpha}{2} \| z \|^2 + \frac{\nu}{2} \| \nabla (\nabla u - \nabla z) \|^2 + \omega (u - g)^2 \]

where \( \Omega \subset \mathbb{R}^2 \) is a rectangular domain and \( g \in L^\infty(\Omega) \) is a given image. Here \( \alpha, \beta, \nu \) are positive parameters (\( 2\beta \geq \alpha \geq \beta \)) and the terms depending on \( \epsilon \) are infinitesimals.

Ambrosio-Faina-March discrete model

In [2] a discrete approximation of BZ functional is proposed; this functional method for image segmentation: this approach leads to a minimization of a non-convex objective energy functional.

OPARBCDA method

Algorithm 1 OPARBCDA

1. **Step 0**: Given \( x^0 \), the partitions \( \{ T_1, \ldots, T_p \} \) and \( \{ S_1, \ldots, S_p \} \) of \( \Omega \) such that \( S_j \cap T_j = \emptyset, j = 1, \ldots, p \) and \( \{ \theta_i \} \), such that \( \theta_i < \theta_j \leq \theta_j + \kappa, \kappa > 0 \) and an exit tolerance \( \theta_{exit}, \kappa \geq 0 \).

2. **Step 1**: for \( j = 1, \ldots, p \)

   - if \( \nabla_{\gamma_j} F(x^j) \neq 0 \) then
     - compute \( y^j = (x^j, z^j, x^j, \ldots, x^j) \):
       - (a) set \( z^{j+1} = z^j + \kappa \cdot \nabla_{\gamma_j} F(x^j) \),
       - (b) repeat:
         - \( k = k + 1 \), compute \( x^{j+1}_k \) by a step of BCDCA; extract \( x^{j+1}_k \),
         - \( x^{j+1}_k - x^j \neq 0 \),
         - \( x^{j+1}_k = x^j \),
       - until \( \| \nabla_{\gamma_j} F(x^j) \| \leq \theta_{exit} \|
     - \( \gamma^j \),
   - else end

3. **Step 2**: define the new iterate \( x^{j+1} \):

   - 2.1 compute \( F_j(y) \) where \( y = (x_1, x_2, \ldots, x_p) \)
   - 2.2 update \( x^{j+1} = \text{argmin}_y \{ F_j(y)^2 \} \)

In this work we are concerned with Blake–Zissermann [1] (BZ) variational method for image segmentation: this approach leads to a minimization of a non-convex objective energy functional. Very often, the segmentation of large-size gridded data is addressed via tiling procedure: additional specific post-processing on tile boundaries may be needed in order to reduce the effect of the subdivision. We aim to show that a simple tiling strategy enables us to treat large images even in a commodity multicore CPU, with no need of specific post-processing on tile junctions.

BCDA sequential approach

In [2] a discrete approximation of BZ functional is proposed; this functional is not globally convex, but it is quadratic and strongly convex w.r.t. each block of variables \( (s, z, u) \).

- when fixing \( u \):
  \[
  F_k(s, z, u) = -\ell \cdot \frac{1}{2} \| s \|^2 + \| z \|^{2} - \frac{\nu}{2} \| \nabla (\nabla u - \nabla z) \|^{2} + \omega \| u - g \|^{2},
  \]

- when fixing \( s, z \):
  \[
  F_k(s, u, z) = -\ell \cdot \frac{1}{2} \| s \|^2 + \| z \|^2 - \frac{\nu}{2} \| \nabla (\nabla u - \nabla z) \|^2 + \omega \| u - g \|^2.
  \]

OPARBCDA parallel approach

In view of the local features of \( F(s, z, u) \), a natural way to address its minimization is to split the image into \( p \) tiles \( T_j, j = 1, \ldots, p \), inducing a partition of the variables \( s, z, u \) into \( p \) blocks \( x_1, x_2, \ldots, x_p \), with \( x_j = (s_j, z_j, u_j) \), \( j = 1, \ldots, p \). In order to avoid side effects on the tile junctions, we enlarge each \( T_j \) of size \( T_j \times (\lambda T_j + 2\sigma) \), where \( \lambda \) is the number of overlapping pixels and \( \sigma = 3\sigma \) (\( \lambda \approx 2 \)).

Parallel implementation

At each outer iteration, Step 1 consists of a number of independent tasks that can be concurrently solved. Mapper/workers pattern encourages run-time distribution of independent tasks among POSIX threads: mutex-protected queues collect both task input and output results: a number \( C \) of computational threads (workers) is initialized and put on wait on a shared task queue, while a monitor thread (master) is responsible to extract, for each subproblem \( j \), initial data \( w_j \) from current solution \( x^j \) and collect subproblems computed solutions. As regards Step 2.2, OpenMP compiler directive omp parallel for is used for evaluation of \( F_j(\gamma) \).

Numerical evaluation

We considered a 2020 × 2020 image and compared the solution \( x^* = (u^*, s^*, z^*) \) obtained by BCDA on the whole dataset and the one \( x^* = (u^*, s^*, z^*) \) computed by OPARBCDA, splitting the image into 8 × 8 and 16 × 16 tiles. BCDA is stopped when the relative difference of \( F \) at two successive iterates is less than 1e-03, while OPARBCDA exits when the current value of \( F \) is less or equal than the minimum achieved by BCDA. We performed runs with up to 15 workers plus one monitor, while for Step 2 parallelization we set the number of OpenMP threads equal to \( C+1 \): this approach would ensure a total number of active threads equal to \( C+1 \) at each parallelized step of the algorithm.

Accuracy on tile junctions

Entries of central portions of \( |z^j - z^{j'}| > 0.01 \) with \( t = 8 \times 8, \nu = 0 \) and \( \nu = 4 \).

OPARBCDA parallel approach

In view of the local features of \( F(s, z, u) \), a natural way to address its minimization is to split the image into \( p \) tiles \( T_j, j = 1, \ldots, p \), inducing a partition of the variables \( s, z, u \) into \( p \) blocks \( x_1, x_2, \ldots, x_p \), with \( x_j = (s_j, z_j, u_j) \), \( j = 1, \ldots, p \). In order to avoid side effects on the tile junctions, we enlarge each \( T_j \) of size \( T_j \times (\lambda T_j + 2\sigma) \), where \( \lambda \) is the number of overlapping pixels and \( \sigma = 3\sigma \) (\( \lambda \approx 2 \)).

Parallel implementation

At each outer iteration, Step 1 consists of a number of independent tasks that can be concurrently solved. Mapper/workers pattern ensures run-time distribution of independent tasks among POSIX threads: mutex-protected queues collect both task input and output results: a number \( C \) of computational threads (workers) is initialized and put on wait on a shared task queue, while a monitor thread (master) is responsible to extract, for each subproblem \( j \), initial data \( w_j \) from current solution \( x^j \) and collect subproblems computed solutions. As regards Step 2.2, OpenMP compiler directive omp parallel for is used for evaluation of \( F_j(\gamma) \).

References and Acknowledgements


This research was supported by: INDAM-GNCS2016, FIRB2012 grant RBFR13UMAC.