Concurrent Computation of Differential Morphological Profiles for Remote Sensing

Michael H. F. Wilkinson\textsuperscript{1}, Pierre Soille\textsuperscript{2}, and Georgios K. Ouzounis\textsuperscript{2}

\textsuperscript{1} Johann Bernoulli Institute, University of Groningen, The Netherlands
m.h.f.wilkinson@rug.nl

\textsuperscript{2} Global Security and Crisis Management Unit, ISFerea, Inst. for the Protection and Security of the Citizen (IPSC), European Commission, Joint Research Center, Ispra, Italy
\{georgios.ouzounis, pierre.soille\}@jrc.ec.europa.eu

Abstract In this paper we provide an efficient parallel algorithm for reconstruction from markers, and multi-scale analysis through differential morphological profiles, which are top-hat scale spaces based on openings and closings by reconstruction. On eight cores a speed increase of 6.26 over naive computation is achieved.

Keywords Differential morphological profile, concurrent computation.

1 Introduction

The Differential Morphological Profile (DMP) \cite{1} is a feature descriptor that often finds usage in the classification of man-made structures in very high resolution (VHR) satellite image analysis. It is based on multiple computations of openings and closings by reconstruction, at an increasing sequence of scales. Due to the large data bulk acquired and the need for rapid analysis in cases of disaster response, parallel computation is essential, but is not readily achieved \cite{4}. In this paper we provide a parallel algorithm for the DMP, based on the algorithm for attribute filters in \cite{5}, which is easily extended to multi-scale computation. The contribution is twofold: (i) we parallelize reconstruction from markers, and (ii) we increase speed by building and interim data structure known as the Max-Tree \cite{2} once, and compute multiple reconstructions from it. On eight cores a total speed increase with respect to naive computation of 6.26 times is achieved.

2 Differential Morphological Profiles

A differential morphological profile (DMP) \cite{1} is a combined top-hat and bottom-hat scale space based on openings and closings by reconstruction. Focusing on the top-hat scale space, at each point \(x\) we can define an \(I - 1\) long differential opening profile \(\Delta^H(\gamma^\rho(f))(x)\), with an entry for each scale \(\lambda_i\) given by:

\[
\Delta^H(\gamma^\rho(f))(x) = -(\gamma^\rho_{\lambda_i}(f) - \gamma^\rho_{\lambda_{i-1}}(f))(x),
\]

in which \(\gamma^\rho_{\lambda_i}\) is the opening by reconstruction with a disk of radius \(\lambda_i\) and \(\gamma^\rho_{\lambda_0}(f) = f\). The differential closing profile \(\Delta^H(\varphi^\rho(f))(x)\) is defined analogously. The differential morphological profile \(DMP\) of a point \(x\) is the \(2(I - 1)\) long vector given by the concatenation (denoted by \(\sqcup\)) of \(\Delta^H(\gamma^\rho(f))(x)\) and \(\Delta^H(\varphi^\rho(f))(x)\), i.e.:

\[
DMP(x) = \Delta^H(\gamma^\rho(f))(x) \sqcup \Delta^H(\varphi^\rho(f))(x).
\]

The two derivative profiles are also referred to as the \textit{opening instance} and \textit{closing instance} of the DMP respectively. An example DMP is shown in Fig. 2. The DMP vector field has also been used to define a segmentation scheme based on the morphological characteristics as in \cite{1}, which is often used for the extraction of build-up in urban areas.
3 Parallel Computation

For the opening part of the DMP we first compute the marker images, by erosions with exact Euclidean disks with the algorithm from [3]. This is parallelized trivially by splitting the image into as many slices as there are processors, and the algorithm is applied to each slice separately.

The second stage entails computing Max-Trees [2]. In a Max-Tree, each node represents a connected component \( X_h(f) \) of grey-scale image \( f \) given by

\[
X_h(f) = \{ x \in E \mid f(x) \geq h \}
\]  

(3)

with \( E \) the image domain. Because each component of \( X_h(f) \) is nested within a larger component \( X_{h'}(f) \) with \( h' < h \), these components form a tree with the image domain as the root. Reconstruction from markers can be computed by building such a Max-Tree, and computing the maximum value of the marker within the image area corresponding to the each node. Once the tree has been built, each node receives the maximum marker level within its bounds as output value, unless it is larger than the original value, in which case the node’s grey level is unchanged. The parallel algorithm builds Max-Trees for \( K \) disjoint sections of the image or volume, and combining the partial Max-Trees into a single one, whilst maintaining the correct marker information [5].

In our implementation, we store the input image in array \( f \), and the marker image, which will double as the output image in array \( out \). As in [5], we store the tree in an array of nodes \( node \) of the same size as the image. Thus each pixel is considered a node in a tree. The parent field of each node contains the index of each pixel’s parent. A separate index \( \perp \) is used to flag the fact that the node is the root of the tree. Only those nodes \( n \) in the final tree, for which the grey value \( f[n] \) differs from the parent grey value \( f[node[n].parent] \) are considered valid Max-Tree nodes. These nodes, and the root node of the tree are referred to as level roots or canonical elements of the components represented in the tree. These level roots can be accessed using a function \( levroot(n) \) which returns the level root of \( n \), and \( Par(n) \) which returns the level root of \( node[n].parent \). We can save memory further by using the \( out \) array to store the maximum marker value within a node. Instead, each node has a boolean field \( valid \), to indicate if it has been reconstructed already.

The building phase of the algorithm is essentially the same as that of attribute filters in [5]. The difference lies in the filtering phase of the algorithm shown in Alg. 1. For each node within our section we descend the tree, moving from level root to level root, until a node \( w \) is found for which either (i) \( node[w].valid \) is true, (ii) \( Par(w) \) has a marker grey value no smaller than its original; or (iii) \( Par(w) = \perp \). In the first case \( w \) has been reconstructed previously, so its output value is correct, in the second case \( w \) is reconstructed to the maximum of \( f[Par(w)] \) and \( out[w] \), unless this is larger than \( f[w] \), in which case \( f[w] \) becomes the reconstructed value. In the final case the
Algorithm 1 Concurrent implementation of the reconstruction phase

procedure MaxTreeReconstr (Vp : Section) =
   for all v ∈ Vp do
      if not node[v].valid then
         w := v;
         while Par(w) ≠ ⊥ ∧ not node[w].valid ∧ f[Par(w)] > out[Par(w)] do
            w := Par(w);
         end;
      if node[w].valid then
         val := out[w]; (* Reconstructed node found *)
      else if Par(w) ≠ ⊥ then
         val := (out[w] ∨ f[Par(w)]) ∧ f[w]; (* w is reconstructed *)
      else val := 0; end; (* marker was empty *)
   end;
   for all u in root path from v to w inclusive do
      if u ∈ Vp then out[u] := val; node[u].valid := true; end;
   end;
end;
end.

marker image was empty, and zero is chosen as the output value. In all cases the reconstruction value of w is propagated up the tree.

We can use this adaptation of [5] to reconstruction from markers to compute the DMP by direct implementation of (2). However, this would entail repeated building of the same Max-Tree. Instead, we can maintain information on multiple markers simultaneously in the Max-Tree. Thus we:

- Compute all markers using parallel erosions and store each scale i in an image out[i].
- Compute a single Max-Tree in parallel, maintaining the maximum marker for all scales i for each node n in out[i][n].
- For all scales i
  - Set all nodes to invalid indicating that they have not been filtered yet (in parallel)
  - Compute reconstruction (in parallel) for scale i, storing the result in out[i].
- Compute differences between scales (in parallel)
- Output opening instance of DMP

Between the different stages a barier is needed for synchronization of the threads, and within the loop over the scales a barier is needed at each iteration. The closing instance is computed by inverting the image and repeating the first stage. For reasons of space the algorithm is not given in more detail, but the code is available upon request.

4 Experiments

The proposed algorithm was tested on a very high resolution (VHR) satellite image of Legaspi (Phillipines). The image is a panchromatic tile of 8-bit resolution, it consists of 5998 × 5998 elements (approx. 36MB), and it is of 0.6m × 0.6m spatial resolution. The original was acquired on the 7th of November 2005 and is courtesy of the original Quickbird Imagery @ DigitalGlobe, Inc; 2005 Distributed by Eurimage. A resampled version of the original is shown in Fig. 2(a).
Figure 2: Timings: Wall-clock time (left) and speed up (right) as a function of number of threads.

The machine used for timing both processes was a 2 Intel Xeon Quad-Core X5470 (3.3GHz, 1.333MHz FSB, 12MB) with a total of 16GB DDR2 Quad Channel FBD system memory.

Computing a nine scale DMP using nine separate sequential reconstructions takes 250.1 s on this machine, of which 132.3 s entails computation of the markers. By contrast the proposed algorithm takes 171.4 s on a single thread. Because the gain is purely in the reconstruction phase, this means a threefold increase in speed of reconstruction (118.1 vs 39.4 s).

The timing results are shown in Fig. 2. At 4 threads the computation time drops to 53.48 s (speed up 3.2): 38.92 s for marker computation (speed-up 3.4), and 14.56 s for reconstruction (speed up 2.7). The overall efficiency is 80%. At 8 threads the speed up is less impressive: 5.1 for the markers, 2.8 for reconstruction, 4.3 total. Overall speed gain with respect to separate computation of each reconstruction is 6.26 overall (5.1 for the markers, and 8.5 for the reconstructions).

5 Conclusions

In this paper we present an algorithm for parallel computation of both reconstruction from markers, and the DMP. Speed-up up to 4 threads on the test machine was good, beyond four the efficiency dropped off rapidly, possibly due to the FSB architecture which limits memory bandwidth. To test this, we will perform tests on machines with different memory access, such as HyperTransport from AMD and QuickPath from Intel. We should also look at reducing the number of barriers in the algorithm, to increase efficiency. As it stands, we obtain significant gains in speed with respect to the original, sequential approach.

References


