Blockwise Algorithms

Martin Bidlingmaier

October 21, 2014

Outline

Introduction

Connected Components Labeling
- Definitions
- MapReduce
- Implementation
- Usage

Watershed Transform
- Definitions
- Implementation
- Usage

Introduction

The Problem

- Large images do not fit in RAM
- Algorithms have to use multi-core CPUs

Introduction

The Problem

- Large images do not fit in RAM
- Algorithms have to use multi-core CPUs

ChunkedArray

- Holds images divided into smaller blocks
- Only loads blocks currently required, caches them

Introduction

The Problem

- Large images do not fit in RAM
- Algorithms have to use multi-core CPUs

ChunkedArray

- Holds images divided into smaller blocks
- Only loads blocks currently required, caches them

Needs adjusted algorithms to be efficient

Connected Components Labeling

Definitions

Definition

Let $X \subseteq \mathbb{Z}^n$, I an image on X.
Let $P(I(x), I(y))$ be a symmetric predicate defined for each adjacent pair of coordinates (x, y) in X.
Define an undirected graph $G = (X, E)$ by setting

$$(x, y) \in E \Leftrightarrow x \textit{ is adjacent to } y \wedge P(I(x), I(y)).$$

A labeling of I according to P is an image J on X such that

$$\forall x, y \in X : J(x) = J(y) \Leftrightarrow x \rightsquigarrow y \textit{ in } G.$$

Connected Components Labeling

MapReduce

MapReduce

1. Divide problem into smaller subproblems
2. Map a function on subproblems (possibly in parallel)
3. Reduce results to a global result

Connected Components Labeling

MapReduce

MapReduce

1. Divide problem into smaller subproblems
2. Map a function on subproblems (possibly in parallel)
3. Reduce results to a global result

MapReduce on ChunkedArrays

1. Image is already stored in separate chunks
2. Map algorithm for MultiArrays on every chunk
3. Reduce subresults to global result

Connected Components Labeling

Implementation - Map Stage

Apply map function

- Iterate over chunks with `ChunkIterator`
- Use `labelMultiArray` to create a local labeling for each chunk
- Save number of local labels assigned for each chunk

Connected Components Labeling

Implementation - Reduce Stage

Goal:

Merge local labels to global labels

Connected Components Labeling

Implementation - Reduce Stage

Goal:

Merge local labels to global labels

Unique global ids for local labels

- Calculate an `id_offset` for each chunk such that `id_offset + local_label_id` yields globally unique label ids

Connected Components Labeling

Implementation - Reduce Stage

Goal:

Merge local labels to global labels

Unique global ids for local labels

- Calculate an `id_offset` for each chunk such that `id_offset + local_label_id` yields globally unique label ids

Merge labels

- Union-find data structure for global label ids
- Iterate over all adjacent chunks with `GridGraph`
- Iterate over adjacent pixels in different chunks with `visitBorder`
- Merge two pixels' global labels if they satisfy the predicate
- Replace local labels by global labels (optional)

Blockwise Labeling

Usage

```
#include <vigra/blockwise_labeling.hxx>
using namespace vigra;

int main() {
  ChunkedArray<4. int>& data = ...
  ChunkedArray<4, int>& labels = ...
  LabelOptions options;
  options.neighborhood(IndirectNeighborhood)
         .background(3);
  labelMultiArrayBlockwise(data,
      labels, options);
   ...
}
```

Watershed Transform

Definitions

Definition

Let I be a grayscale image on $X \subseteq \mathbb{Z}^n$. I can be regarded as a topographic relief by identifying darkness with height for every pixel.

A drop of water put on a pixel will flow down the steepest slope until it stops in a minimum. A watershed labeling according to the drop of water principle is an image J on X such that

$$\forall x, y \in X : J(x) = J(y) \Leftrightarrow \textit{drops of water put on } I \textit{ at positions } x \textit{ and } y \\ \textit{come to a halt in the same minimum}$$

Watershed Transform

Definitions

Definition

Let I be a grayscale image on $X \subseteq \mathbb{Z}^n$. I can be regarded as a topographic relief by identifying darkness with height for every pixel.

A drop of water put on a pixel will flow down the steepest slope until it stops in a minimum. A watershed labeling according to the drop of water principle is an image J on X such that

$$\forall x, y \in X : J(x) = J(y) \Leftrightarrow \begin{array}{l} \textit{drops of water put on } I \textit{ at positions } x \textit{ and } y \\ \textit{come to a halt in the same minimum} \end{array}$$

Problem: non-lower-complete images

Watershed Transform

Definitions

A watershed labeling can be reduced to a connected components labeling problem with the predicate

$$\begin{aligned} P(x,y) \Leftrightarrow\ & \textit{x is the lowest neighbor of y} \vee \\ & \textit{y is the lowest neighbor of x} \vee \\ & \textit{neither x nor y has a strictly lower neighbor} \end{aligned}$$

Watershed Transform

Definitions

A watershed labeling can be reduced to a connected components labeling problem with the predicate

$$\begin{aligned} P(x,y) \Leftrightarrow\ & \textit{x is the lowest neighbor of y} \vee \\ & \textit{y is the lowest neighbor of x} \vee \\ & \textit{neither x nor y has a strictly lower neighbor} \end{aligned}$$

To decide $P(x,y)$, all neighbors of x and y have to be considered – bad for a blockwise algorithm (pixels on chunk borders)

Blockwise Watershed Transform

Implementation

Solution:

- Checkout blocks slightly larger than chunks that overlap adjacent chunks by one pixel
- Save relative coordinate of lowest neighbor for each pixel in a temporary array
- Use only temporary array to decide predicate and label according to it
- Write operations only within the actual chunk size
  $\Rightarrow$ parallelizable

Blockwise Watershed Transform

Usage

```
#include <vigra/blockwise_watershed.hxx>
using namespace vigra;

int main() {
  ChunkedArray<4. int>& data = ...
  ChunkedArray<4, int>& labels = ...
  unionFindWatershedsBlockwise(data, labels,
      IndirectNeighborhood);
   ...
}
```

Thank you!