Parallelization of Burn Scar Mapping algorithms

Undergraduate Thesis

Nikolaos Kanakaris

Department of Informatics and Telematics
Harokopio University of Athens

Burn Scar Mapping

Problem

Nowadays, there are more and more fire breakouts, in forests, throughout the world.

What do we need?

We need a way to quantify the forest fires per year.

Solution

Burn Scar Mapping (BSM) identifies and points out burnt areas using satellite images from various spectral bands.

Related work

Nasa

Nasa has implemented BSM tools. These tools use a variety of data such as:

- Data from the MODIS sensor
- Data from Landsat program satellites

National Observatory of Athens

NOA has implemented BSM tools, too (e.g. FireHub). The aim of these tools is to process data collected on Greek territory.

This thesis is related to NOA's BSM tool, which utilizes Landsat's images.

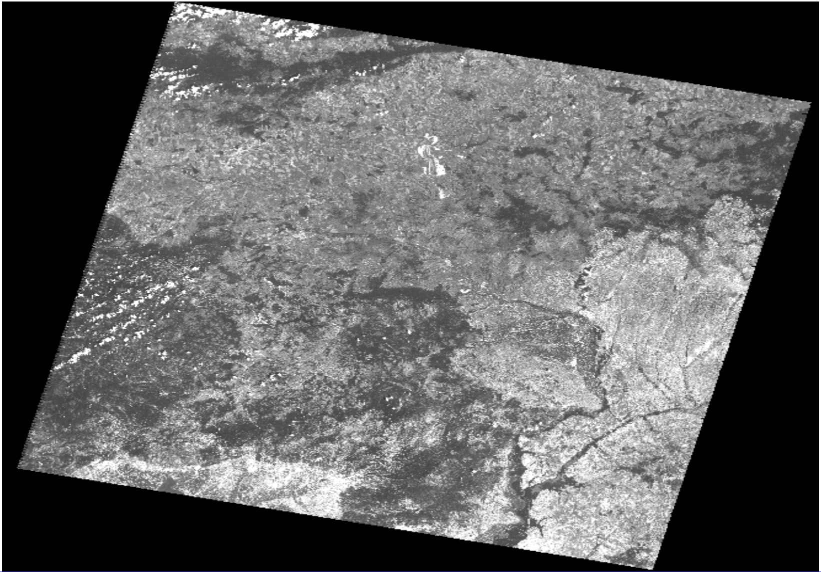
Landsat image

A Landsat image consists of many satellites images. Actually, it is a bundle of sub-images. Usually, a sub-image is captured in the spectral bands 3, 4 or 7.

Details

- It consumes a significant amount of memory (e.g. $328Mb$)
- Each sub-image has more or less $8781x9678$ pixels
- Each band contains different frequencies
- Landsat 7 satellite is used to collect our data

Landsat image
A sub-image example, band 3



NOA's BSM tool
Overview

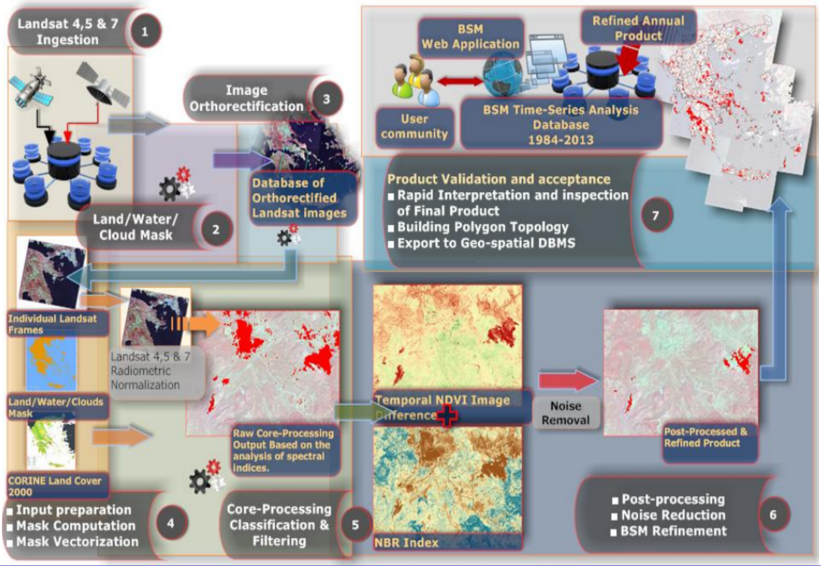
NOA's BSM tool is an automatic image processing chain. It integrates three stages of processing.

Stages

- Pre-processing (orthorectification, cloud/water masking)
- Core BSM (this is where we come in)
- Post-processing (integration, extra refinement using more layers of information; e.g. land-cover maps)

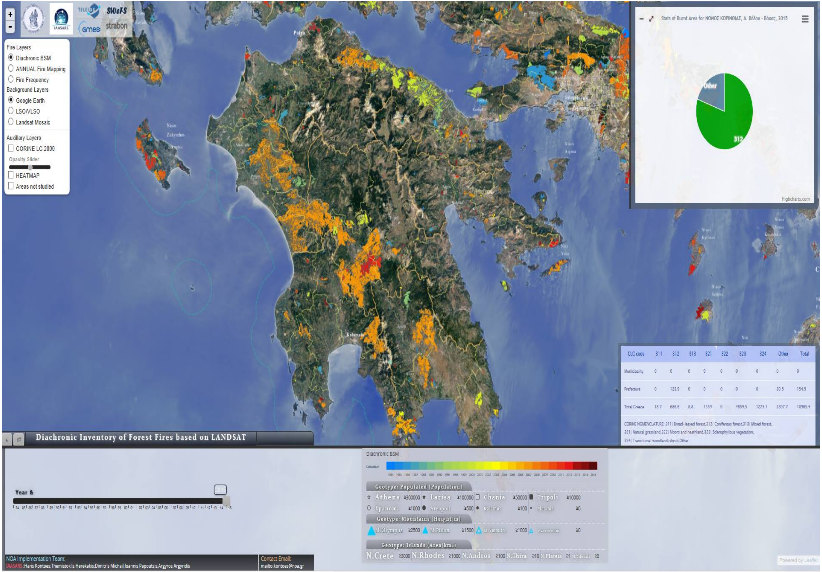
NOA's BSM tool

Overview (source:http://www.earthobservatory.eu/publications/EARSEL2013.pdf)



NOA's BSM tool

Output (source: http://ocean.space.noa.gr/fires)



Core BSM
Overview

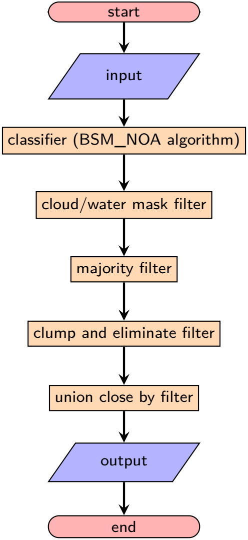
Core BSM applies a chain of filters on the input image/s.

Details

- Python 2.7
- NumPy, gdal, etc
- Sequential implementation
- Terminates in approximately 14.7 minutes (per image)

Core BSM

Chain's structure



Core BSM
Classifier (BSM_NOA algorithm)

The classifier uses indexes like ALBEDO, NDVI vegetation and NBR, in order to verify whether a pixel has been burnt or not.

Details

- $ALBEDO_{snow} = 0.9$, $ALBEDO_{average} \in [0.37, 0.39]$
- $NDVI = \frac{NIR-VIS}{NIR+VIS}$, where $-1 \leq NDVI \leq 1$
- $NBR = \frac{NIR-MIR}{NIR+MIR}$, where $-1 \leq NBR \leq 1$
- ‘Neighborhood operation’ free
- Output: A binary image $(0/1)$

Core BSM
‘Cloud/water mask’ filter

Core BSM also compares the previous binary image with one that describes the cloud/water areas.

Details

- ‘Neighborhood operation’ free
- Removes cloud/water noise (sets wrong burned pixels as unburnt $(1 \to 0)$)
- Output: A binary image $(0/1)$

The 'Majority' filter sets an unburnt pixel as burnt when its nearby burnt pixels are greater than a threshold value.

Details

- 'Neighborhood operation'
- Spatial filter
- Window $3x3$, $5x5$, etc.
- Output: A binary image $(0/1)$

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & \cdots & a_{2,m} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & \cdots & a_{3,m} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & \cdots & a_{4,m} \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & \cdots & a_{5,m} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & a_{n,4} & a_{n,5} & \cdots & a_{n,m} \end{pmatrix}$$

Core BSM

‘Clump and eliminate’ filter

The ‘Clump and eliminate’ filter groups burnt pixels by using the BFS algorithm.

Details

- Each group has an identifier
- Each pixel has a ‘group ID’ as a value
- Eliminates groups with a population less than a threshold

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & \cdots & a_{2,m} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & \cdots & a_{3,m} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & \cdots & a_{4,m} \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & \cdots & a_{5,m} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & a_{n,4} & a_{n,5} & \cdots & a_{n,m} \end{pmatrix} \quad \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & \cdots & a_{2,m} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & \cdots & a_{3,m} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & \cdots & a_{4,m} \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & \cdots & a_{5,m} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & a_{n,4} & a_{n,5} & \cdots & a_{n,m} \end{pmatrix}$$

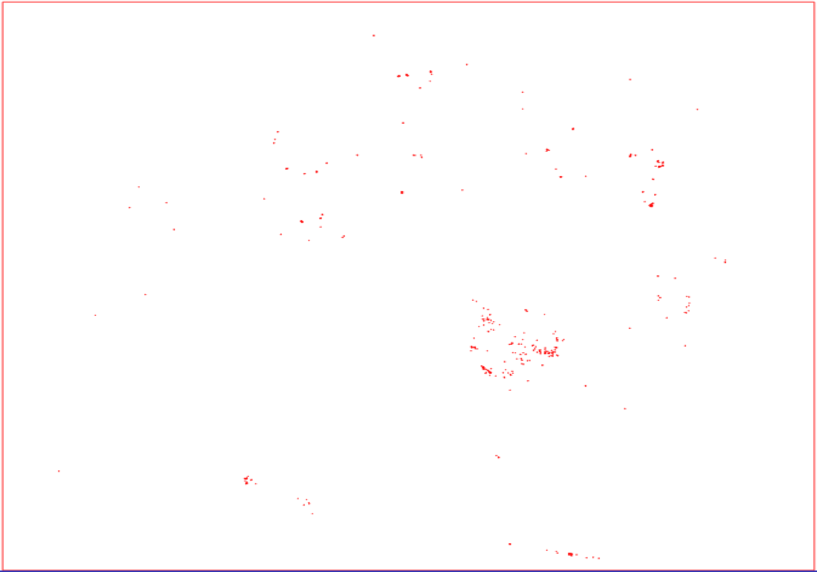
Core BSM
'Union close by' filter

Details

- Merges groups of fires
- Sets as fire the path between them
- Union-Find under the hood

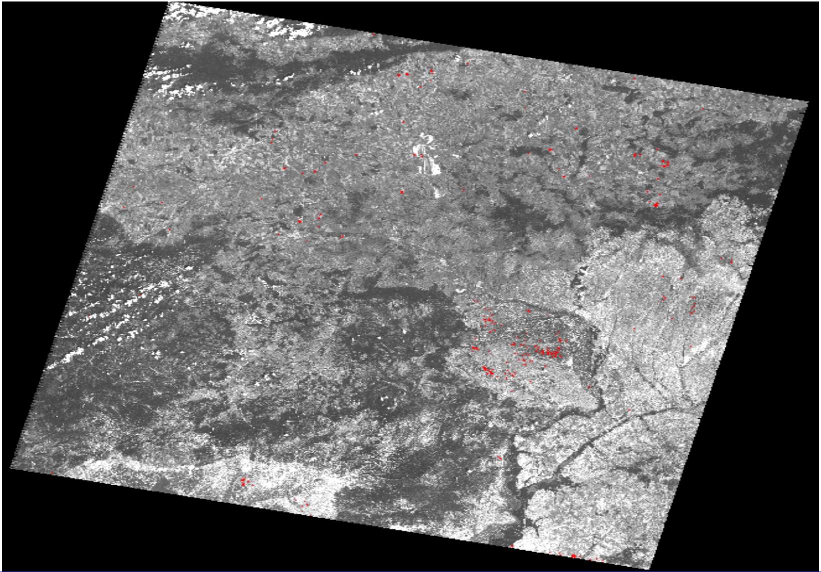
$$
\begin{pmatrix}
a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} & a_{1,7} & a_{1,8} & \cdots & a_{1,m} \\
a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & a_{2,6} & a_{2,7} & a_{2,8} & \cdots & a_{2,m} \\
a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} & a_{3,7} & a_{3,8} & \cdots & a_{3,m} \\
a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} & a_{4,7} & a_{4,8} & \cdots & a_{4,m} \\
a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & a_{5,6} & a_{5,7} & a_{5,8} & \cdots & a_{5,m} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
a_{n,1} & a_{n,2} & a_{n,3} & a_{n,4} & a_{n,5} & a_{n,6} & a_{n,7} & a_{n,8} & \cdots & a_{n,m}
\end{pmatrix}
\begin{pmatrix}
a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} & a_{1,7} & a_{1,8} & \cdots & a_{1,m} \\
a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & a_{2,6} & a_{2,7} & a_{2,8} & \cdots & a_{2,m} \\
a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} & a_{3,7} & a_{3,8} & \cdots & a_{3,m} \\
a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} & a_{4,7} & a_{4,8} & \cdots & a_{4,m} \\
a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & a_{5,6} & a_{5,7} & a_{5,8} & \cdots & a_{5,m} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
a_{n,1} & a_{n,2} & a_{n,3} & a_{n,4} & a_{n,5} & a_{n,6} & a_{n,7} & a_{n,8} & \cdots & a_{n,m}
\end{pmatrix}
$$

Core BSM
Output



Core BSM

Output (multiple layers)



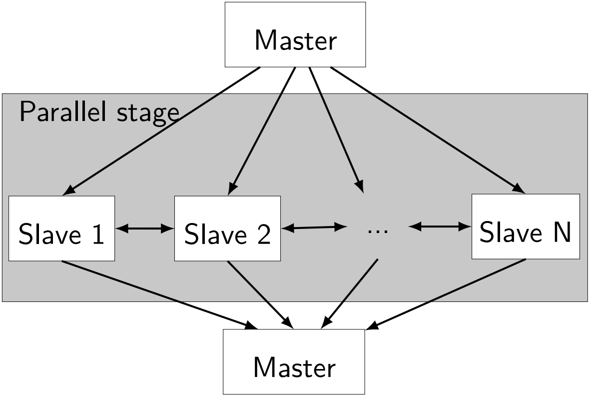
Core BSM (parallel implementation)

Overview

Details

- Message Passing Interface (MPI)
- mpi4py
- Terminates in approximately 37 seconds (per image)

Core BSM (parallel implementation)
Parallel architecture



Core BSM (parallel implementation)

Partitioning

Each process i owns $n_i = n/s + (1 \text{ if } n \mod s > i \text{ else } 0)$ rows. Furthermore, these rows are in range $[a_i, b_i)$, where $a_i = i \cdot n/s + min(i, n \mod s)$ and $b_i = n_i + b_i$.

$$
\begin{pmatrix}
a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\
a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\
a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
a_{12,1} & a_{12,2} & a_{12,3} & \cdots & a_{12,n}
\end{pmatrix}
\xrightarrow{rank\_0}
\begin{pmatrix}
a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\
a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\
a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\
a_{4,1} & a_{4,2} & a_{4,3} & \cdots & a_{4,n}
\end{pmatrix}
$$

$$
\xrightarrow{rank\_1}
\begin{pmatrix}
a_{5,1} & a_{5,2} & a_{5,3} & \cdots & a_{5,n} \\
a_{6,1} & a_{6,2} & a_{6,3} & \cdots & a_{6,n} \\
a_{7,1} & a_{7,2} & a_{7,3} & \cdots & a_{7,n} \\
a_{8,1} & a_{8,2} & a_{8,3} & \cdots & a_{8,n}
\end{pmatrix}
$$

$$
\xrightarrow{rank\_2}
\begin{pmatrix}
a_{9,1} & a_{9,2} & a_{9,3} & \cdots & a_{9,n} \\
a_{10,1} & a_{10,2} & a_{10,3} & \cdots & a_{10,n} \\
a_{11,1} & a_{11,2} & a_{11,3} & \cdots & a_{11,n} \\
a_{12,1} & a_{12,2} & a_{12,3} & \cdots & a_{12,n}
\end{pmatrix}
$$

Core BSM (parallel implementation)
Communication

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ a_{4,1} & a_{4,2} & a_{4,3} & \cdots & a_{4,n} \end{pmatrix}$$

$$\uparrow \downarrow$$

$$\begin{pmatrix} a_{5,1} & a_{5,2} & a_{5,3} & \cdots & a_{5,n} \\ a_{6,1} & a_{6,2} & a_{6,3} & \cdots & a_{6,n} \\ a_{7,1} & a_{7,2} & a_{7,3} & \cdots & a_{7,n} \\ a_{8,1} & a_{8,2} & a_{8,3} & \cdots & a_{8,n} \end{pmatrix}$$

$$\uparrow \downarrow$$

$$\begin{pmatrix} a_{9,1} & a_{9,2} & a_{9,3} & \cdots & a_{9,n} \\ a_{10,1} & a_{10,2} & a_{10,3} & \cdots & a_{10,n} \\ a_{11,1} & a_{11,2} & a_{11,3} & \cdots & a_{11,n} \\ a_{12,1} & a_{12,2} & a_{12,3} & \cdots & a_{12,n} \end{pmatrix}$$

1) Exchanges data

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ a_{4,1} & a_{4,2} & a_{4,3} & \cdots & a_{4,n} \\ a_{5,1} & a_{5,2} & a_{5,3} & \cdots & a_{5,n} \end{pmatrix}$$

$$\begin{pmatrix} a_{4,1} & a_{4,2} & a_{4,3} & \cdots & a_{4,n} \\ a_{5,1} & a_{5,2} & a_{5,3} & \cdots & a_{5,n} \\ a_{6,1} & a_{6,2} & a_{6,3} & \cdots & a_{6,n} \\ a_{7,1} & a_{7,2} & a_{7,3} & \cdots & a_{7,n} \\ a_{8,1} & a_{8,2} & a_{8,3} & \cdots & a_{8,n} \\ a_{9,1} & a_{9,2} & a_{9,3} & \cdots & a_{9,n} \end{pmatrix}$$

$$\begin{pmatrix} a_{8,1} & a_{8,2} & a_{8,3} & \cdots & a_{8,n} \\ a_{9,1} & a_{9,2} & a_{9,3} & \cdots & a_{9,n} \\ a_{10,1} & a_{10,2} & a_{10,3} & \cdots & a_{10,n} \\ a_{11,1} & a_{11,2} & a_{11,3} & \cdots & a_{11,n} \\ a_{12,1} & a_{12,2} & a_{12,3} & \cdots & a_{12,n} \end{pmatrix}$$

2) Executes filter

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n} \\ a_{4,1} & a_{4,2} & a_{4,3} & \cdots & a_{4,n} \\ \cancel{a_{5,1}} & \cancel{a_{5,2}} & \cancel{a_{5,3}} & \cancel{\cdots} & \cancel{a_{5,n}} \end{pmatrix}$$

$$\begin{pmatrix} \cancel{a_{4,1}} & \cancel{a_{4,2}} & \cancel{a_{4,3}} & \cancel{\cdots} & \cancel{a_{4,n}} \\ a_{5,1} & a_{5,2} & a_{5,3} & \cdots & a_{5,n} \\ a_{6,1} & a_{6,2} & a_{6,3} & \cdots & a_{6,n} \\ a_{7,1} & a_{7,2} & a_{7,3} & \cdots & a_{7,n} \\ a_{8,1} & a_{8,2} & a_{8,3} & \cdots & a_{8,n} \\ \cancel{a_{9,1}} & \cancel{a_{9,2}} & \cancel{a_{9,3}} & \cancel{\cdots} & \cancel{a_{9,n}} \end{pmatrix}$$

$$\begin{pmatrix} \cancel{a_{8,1}} & \cancel{a_{8,2}} & \cancel{a_{8,3}} & \cancel{\cdots} & \cancel{a_{8,n}} \\ a_{9,1} & a_{9,2} & a_{9,3} & \cdots & a_{9,n} \\ a_{10,1} & a_{10,2} & a_{10,3} & \cdots & a_{10,n} \\ a_{11,1} & a_{11,2} & a_{11,3} & \cdots & a_{11,n} \\ a_{12,1} & a_{12,2} & a_{12,3} & \cdots & a_{12,n} \end{pmatrix}$$

3) Removes odd data

Core BSM (parallel implementation)
Chain

Classifier (BSM_NOA algorithm)

Free of communication stage

‘Cloud/water mask’ filter

Free of communication stage

‘Majority’ filter

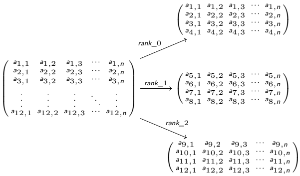
- Communication between neighbor processes, due to spatial filter

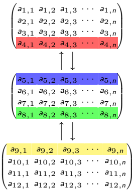
‘Clump and eliminate’ filter

- Mainly, communication between neighbor processes
- Parallel BFS
- Each process sends/receives population of teams (All to all communication)
- Runs BFS algorithm (only for border lines), until there are no changes

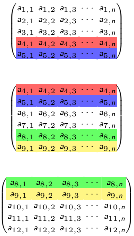
‘Union close by’ filter

- Only Communication between neighbor processes
- Parallel Union-Find
- Our Union-Find needs no all to all communication
- Runs Union-Find (only for border lines), until there are no changes

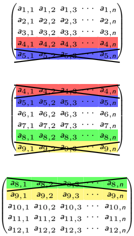




1) Exchanges data

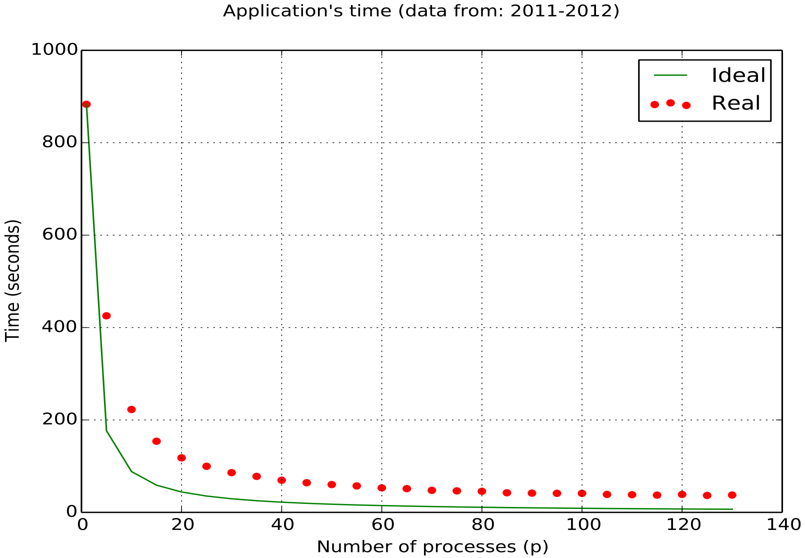


2) Executes filter

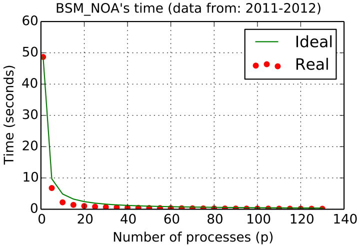


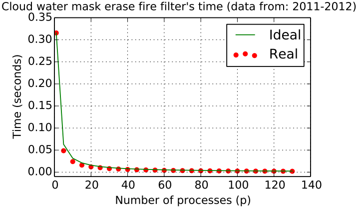
3) Removes odd data

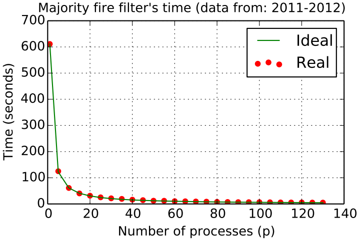
Experimental results
Execution time



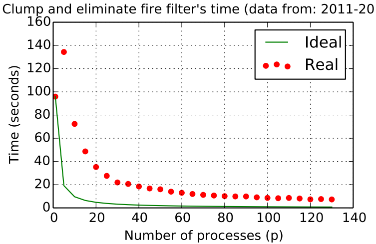
Experimental results
Execution time per filter (1)

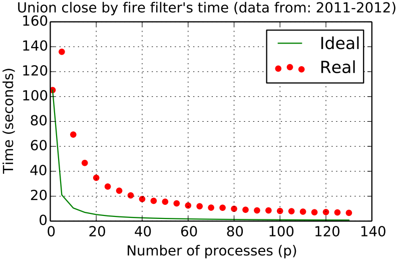




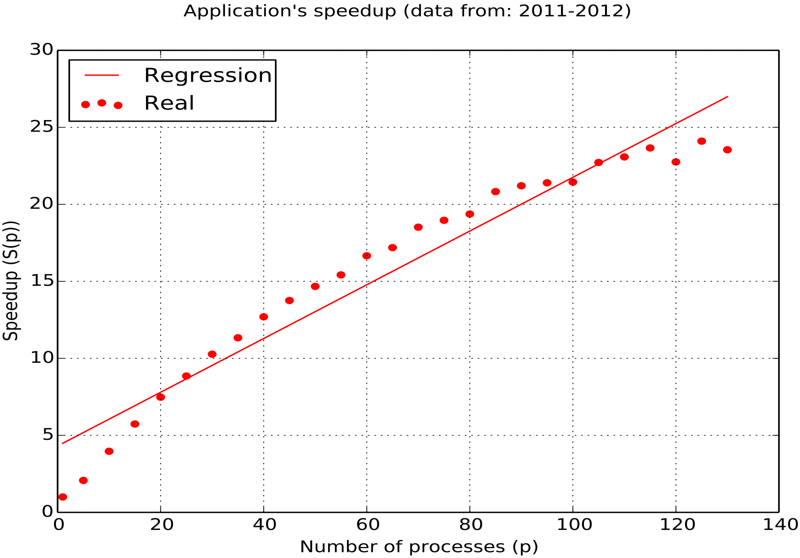


Experimental results
Execution time per filter (2)

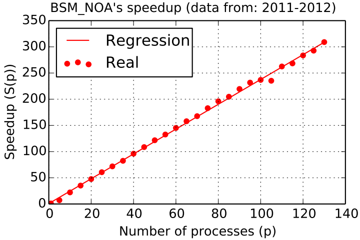


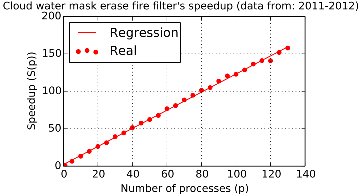


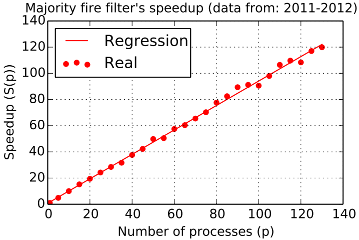
Experimental results
Speedup



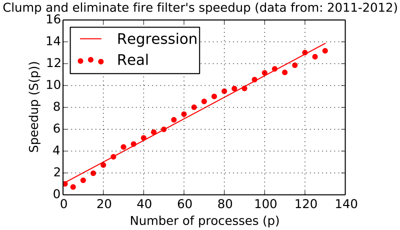
Experimental results
Speedup per filter (1)

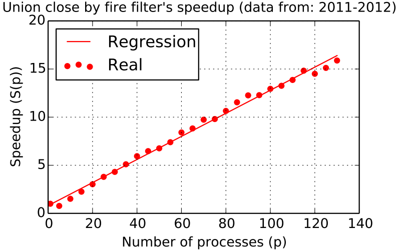






Experimental results
Speedup per filter (2)



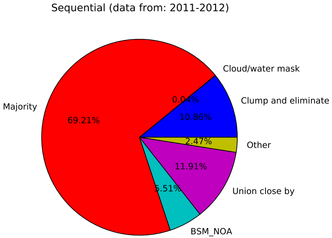


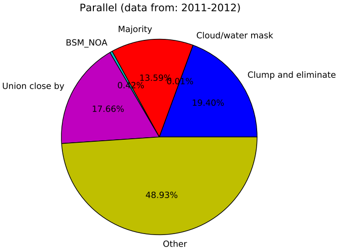
Experimental results

A different scope (132 processes in 33 computers)

	Sequential time	Parallel time	Speedup
BSM_NOA (Classifier)	48.672	0.157	300
Cloud/water mask filter	0.315	0.001	155
Majority filter	611.241	5.098	120
Clump and eliminate filter	95.896	7.275	14
Union close by filter	105.203	6.624	16
Total	883.139	37.510	27

Experimental results
Profiling





Let's improve

Improvements

- Improve 'clump and eliminate' filter (advanced BFS algorithm)
- Improve 'union close by' filter (advanced Union-Find)
- Combine programming languages (C/C++, Python, Cython)
- Combine shared memory and distributed memory systems
- Reduce communication

Any Questions?

The end.
Thank you.

```cpp
#include <iostream>

class Student
{
public:
  Student(const bool yolo_mode): yolo_mode(yolo_mode) {}
  bool is_in_yolo_mode() const { return yolo_mode; }
private:
  bool yolo_mode;
};

int main()
{
  const Student nikos(false);
  if (nikos.is_in_yolo_mode())
    std::cout << "No need for questions!" << std::endl;
  else
    std::cout << "Too bad, I am always in this mode!" << std::endl;
}
```