A GPU-based Architecture for Parallel Image-aware Version Control

Jose Ricardo da Silva Junior Toni Pacheco
Esteban Clua Leonardo Murta

Instituto de Computação

Universidade Federal Fluminense

Schedule

- Introduction
- GPU processing
- VCS operations
- Results
- Conclusion
- Future works

Introduction

- Version control, nowadays, is considered a vital component for supporting professional software development.
  - Mainly based on files and directories.
- Textual artifacts has a well established process.
- Unfortunately, VCS for binary data are not yet well established.

Introduction

So many projects are highly binary data intensive!

Movie industry

Advertising Industry

Game Industry

Normally, has more binary artifacts (sound, 3d models, images) than textual artifacts

Introduction

- Normally, to deal with binary data, two paths are used:
  - Store the binary data as a whole between each modification.
    - Loose semantic information!
    - Requires more storage space.
    - No processing time.
  - Implement algorithms to deal with these binary artifacts.
    - Allows more semantic for the end user.
    - Requires more processing and time!

Storing binary data as a whole

- What has been changed between these two revisions?

Revision 1

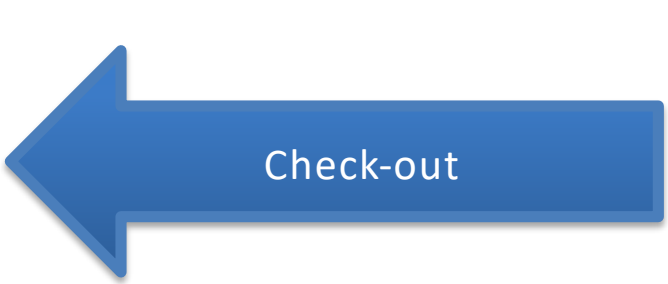
Revision 2

Storing binary data as a whole

- Normally, state-based VCS, such as Git, saves binary data without any delta information.
  - High network traffic in projects that uses a lot of binary artifacts!
  - Slow down operations of check –in and –out!
  - More disk space required.
  - Loose of productivity!

Semantic based algorithms

- Store delta information between binary artifacts.
- Less disk space required.
- Allows more semantic to be presented to user.
- On the other hand, normally requires a lot more processing during check-in and –out operations!



Semantic based algorithms

- In order to process two single images for ***diff*** operation, as example:

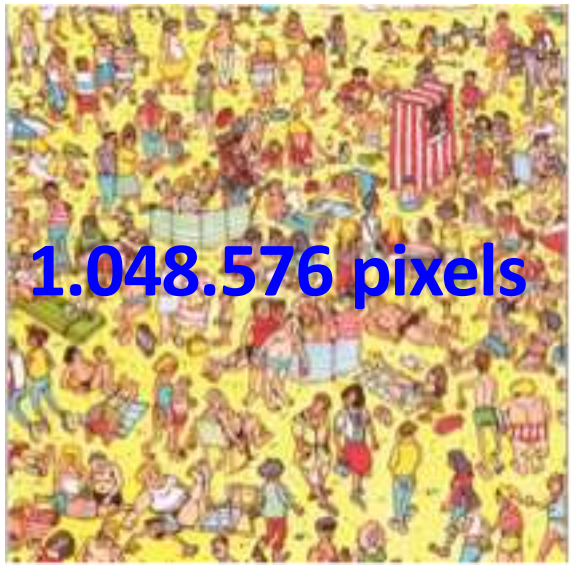
Width: 1024

Height: 1024

Revision 1

Width: 1024

Height: 1024

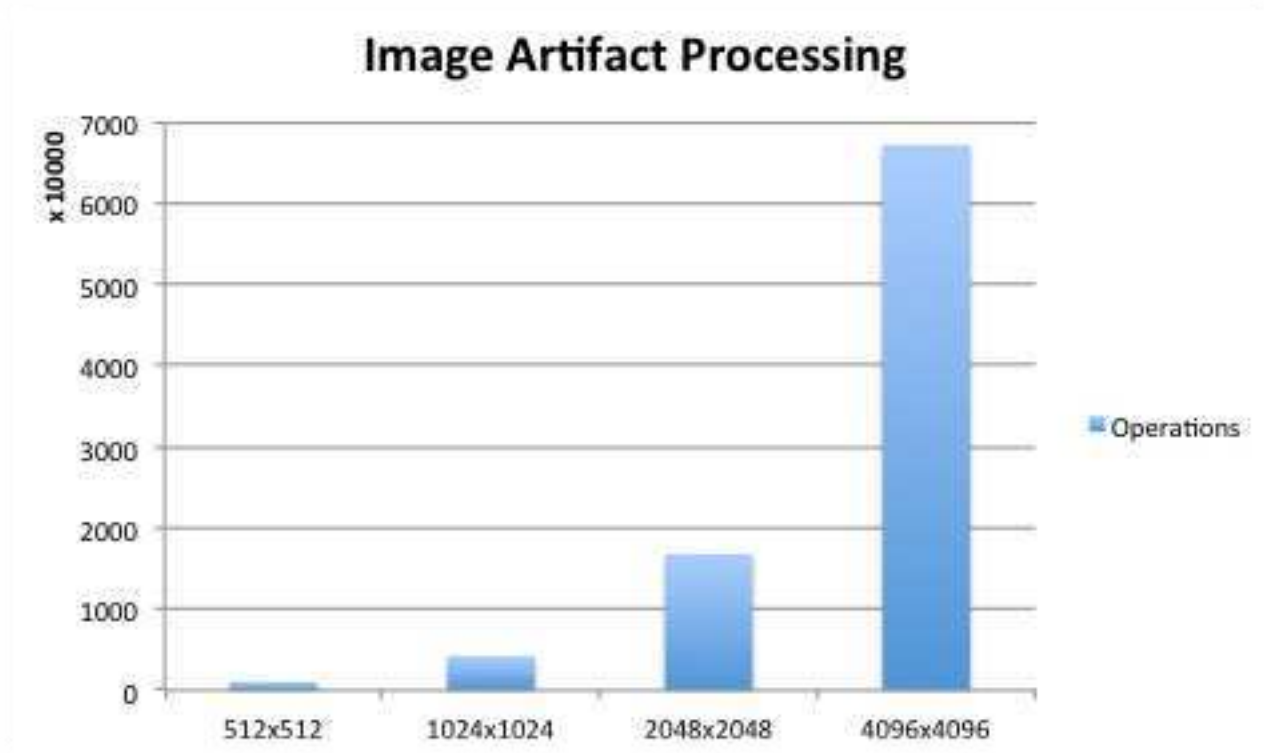


Revision 2

- Processing of ***2.097.152*** elements!

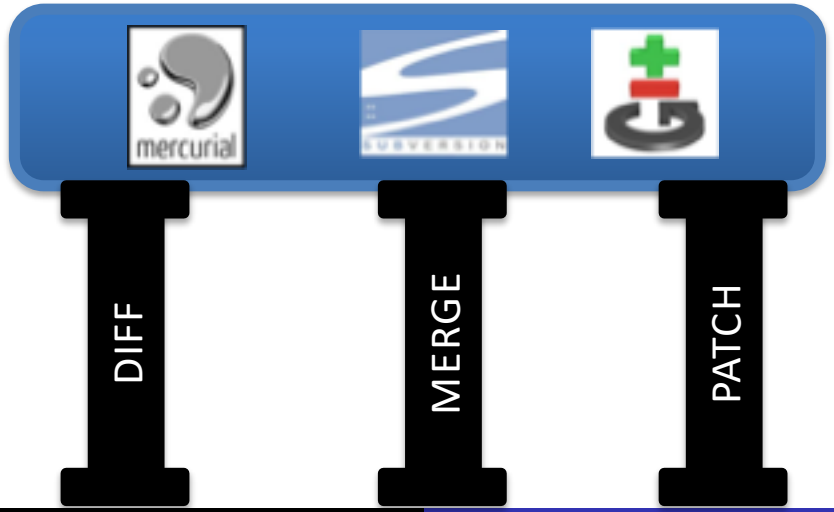
Semantic based algorithms

- Normally, images are composed pixels of three (RGB) or four (RGBA) channels, for **R**ed, **G**reen, **B**lue and **A**lpha, requiring processing each channel, individually during VCS operation.



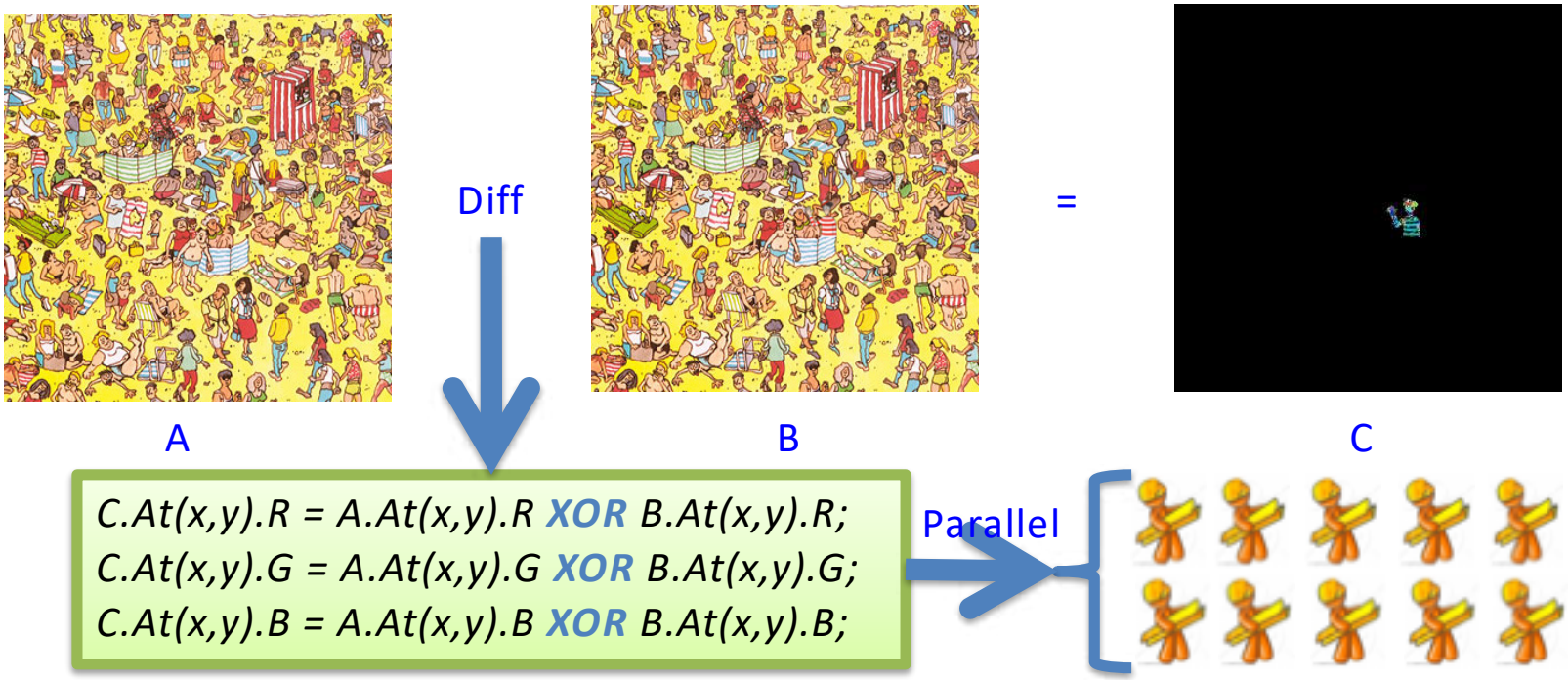
Motivation

- Due to these observations, we are aimed to:
  - Give semantic information for image type artifacts;
  - Process as fast as possible ***diff***, ***patch*** and ***merge*** operations during check-in and –out;
  - Use less space to store delta between revisions.
  - Using GPU due to problem characteristics (data independency).



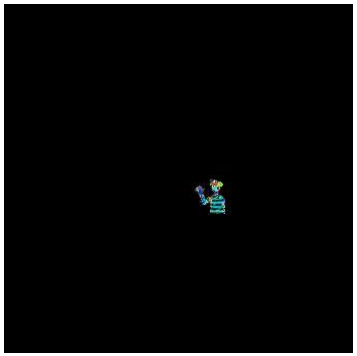
Diff on IMUFF

- Aimed to locate differences between images and save its delta.
  - Uses the ***XOR*** operation on each channel to find its delta.



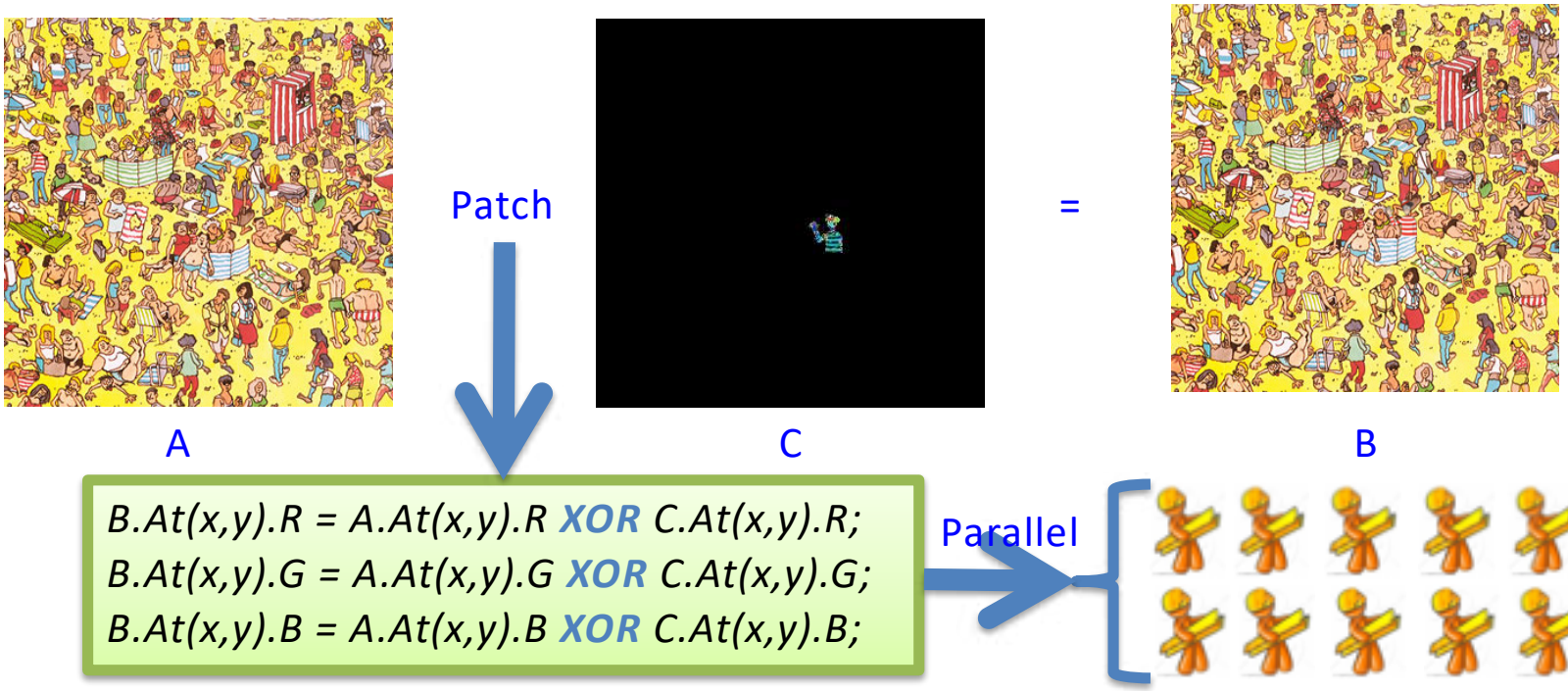
Diff on IMUFF

- As can be observed, most of our delta image is composed of black colors (zeros).
  - After compression, this delta leads to small size, requiring less storage and network bandwidth.
- Usually, small deltas are expected between two consecutives versions.
- Gives the user a high semantic information of the modification.



Patch on IMUFF

- Aimed to reconstruct others revisions.
  - Like ***diff***, uses the XOR operation on each channel to reconstruct a revision.



Patch on IMUFF

- Some properties of XOR operation in IMUFF:

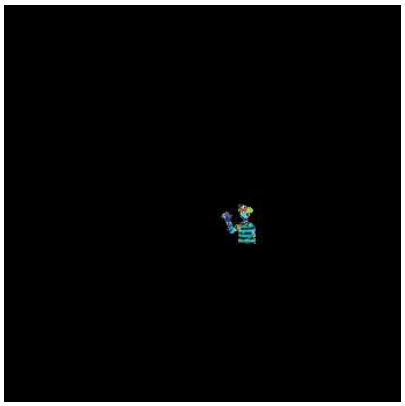
patch(A,C) = A if C is an empty delta

patch (A,C) = patch(C,A) = B

patch (A,C) = B and patch(B,C) = A

B

Patch



C

=

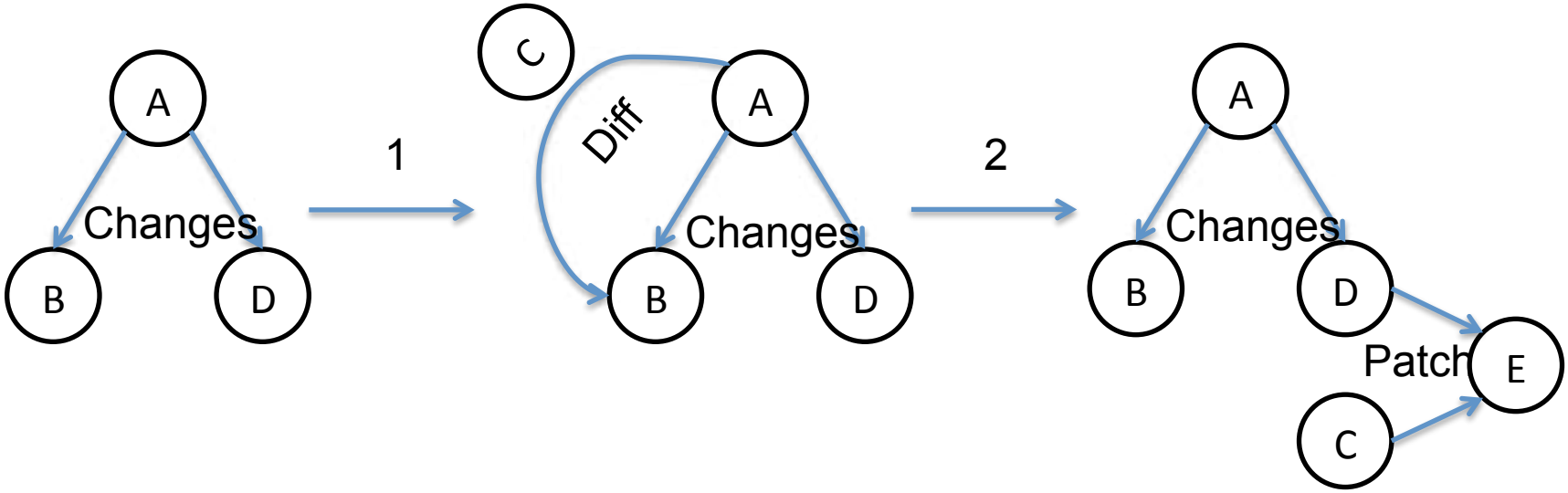
A

```
B.At(x,y).R = A.At(x,y).R XOR C.At(x,y).R;
B.At(x,y).G = A.At(x,y).G XOR C.At(x,y).G;
B.At(x,y).B = A.At(x,y).B XOR C.At(x,y).B;
```

Parallel

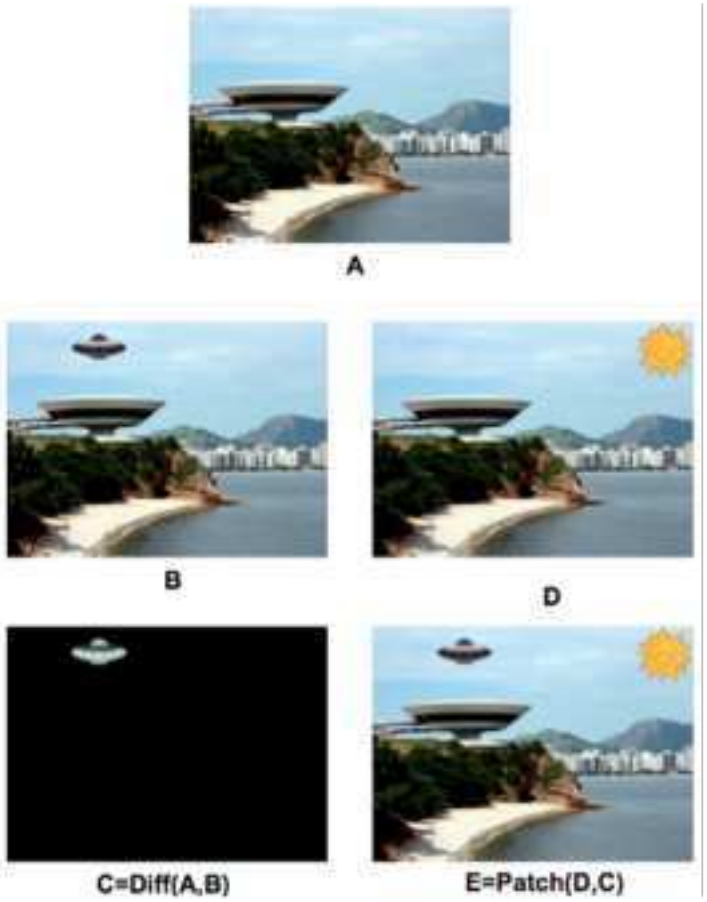
Merge on IMUFF

- Performed to conciliate two revisions created in parallel.
- Uses the previously ***diff*** and ***patch*** operations.

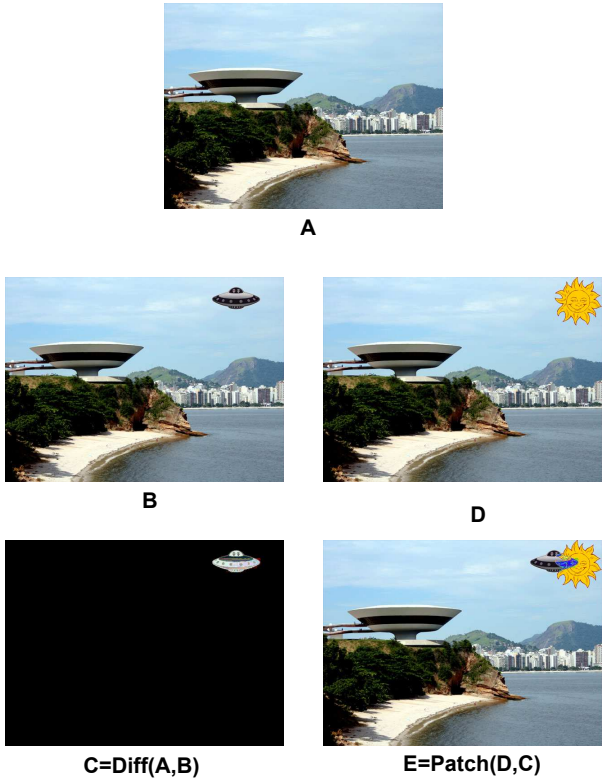


Merge on IMUFF

- Performed to conciliates two revisions created in parallel.

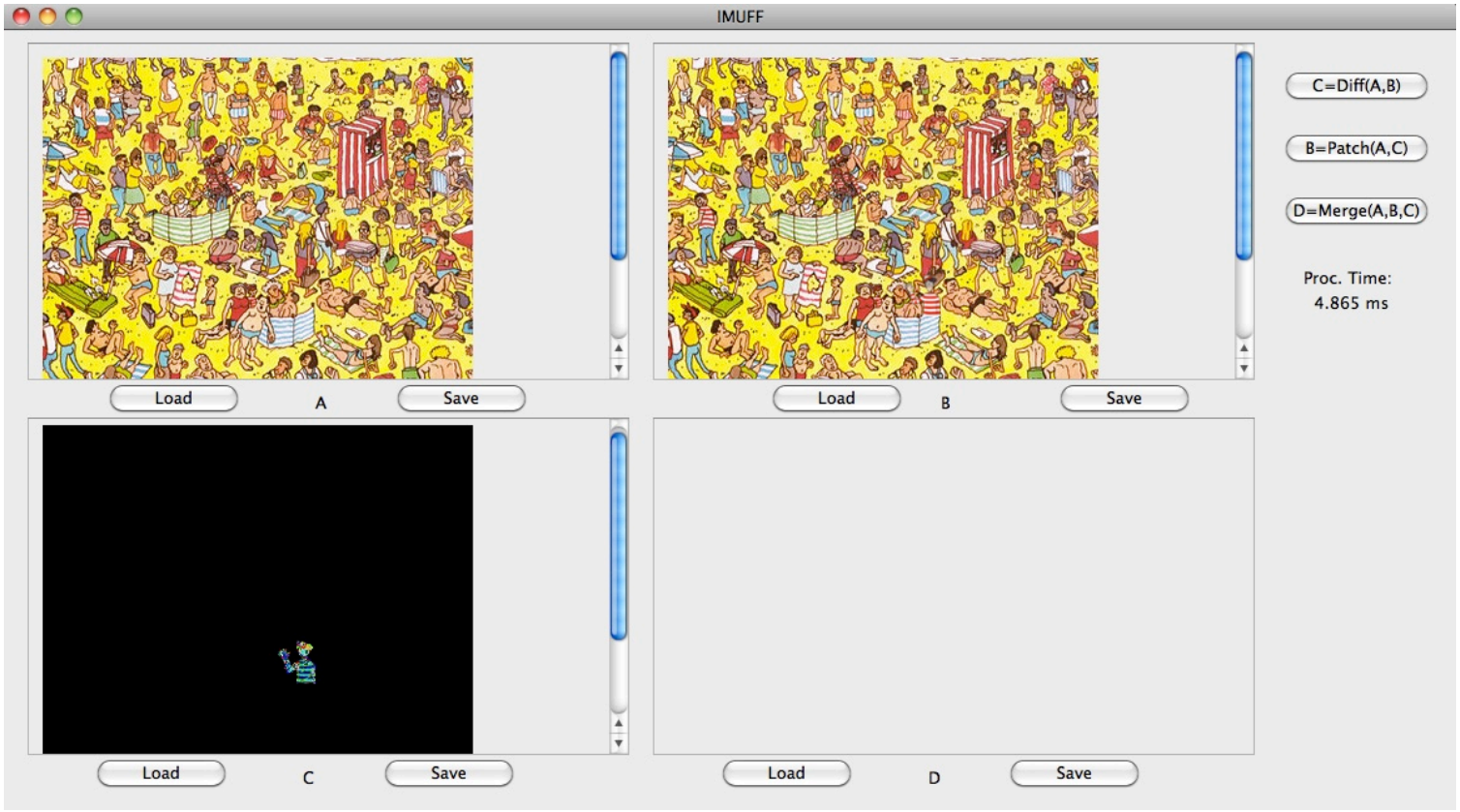


In case the same image area are changed, a conflict in generated, like a common line based VCS.



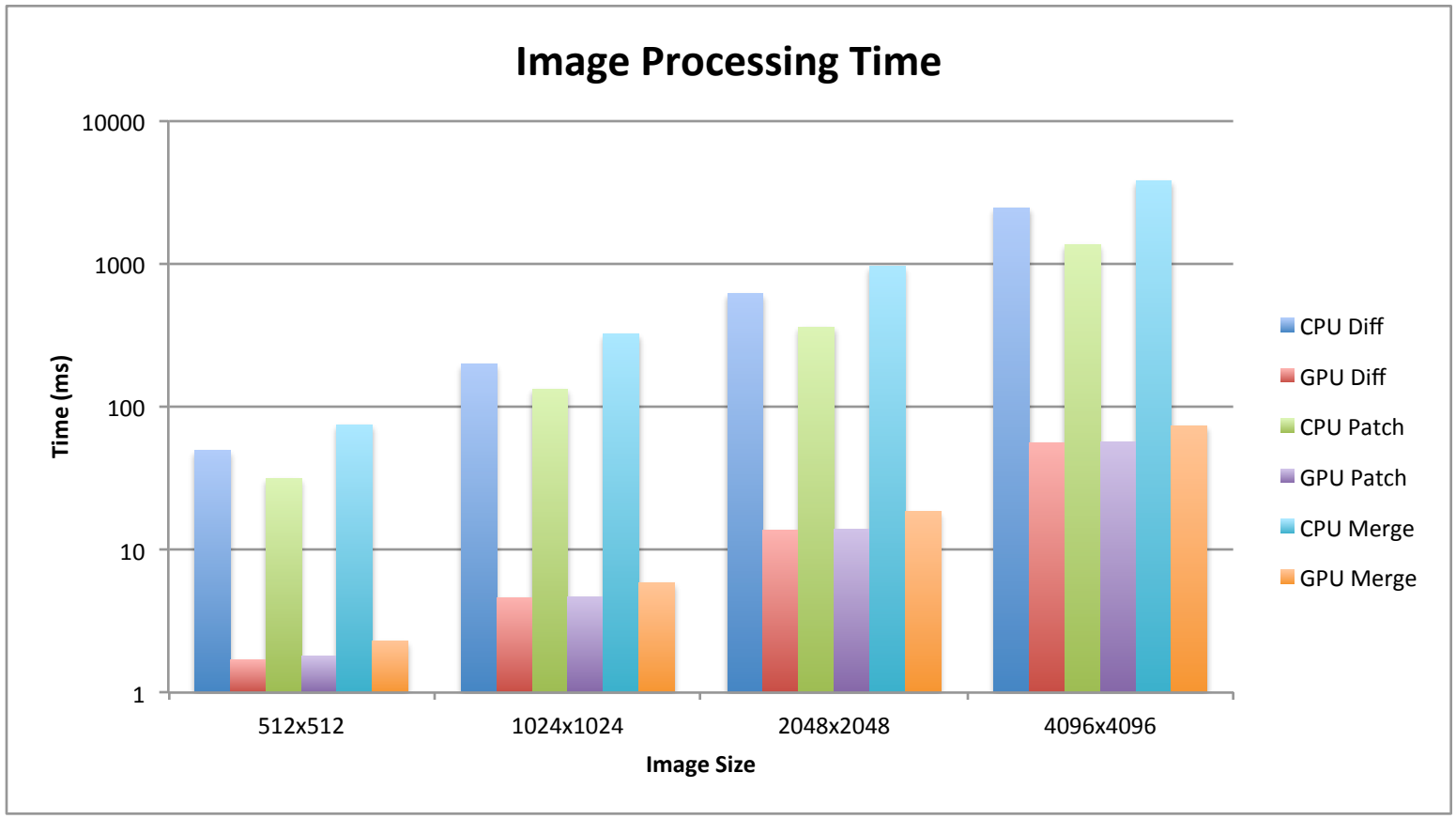
Results

- In order to perform these operations on IMUFF, a GUI is freely available at http://josericardojunior.com/imuff/.



Results

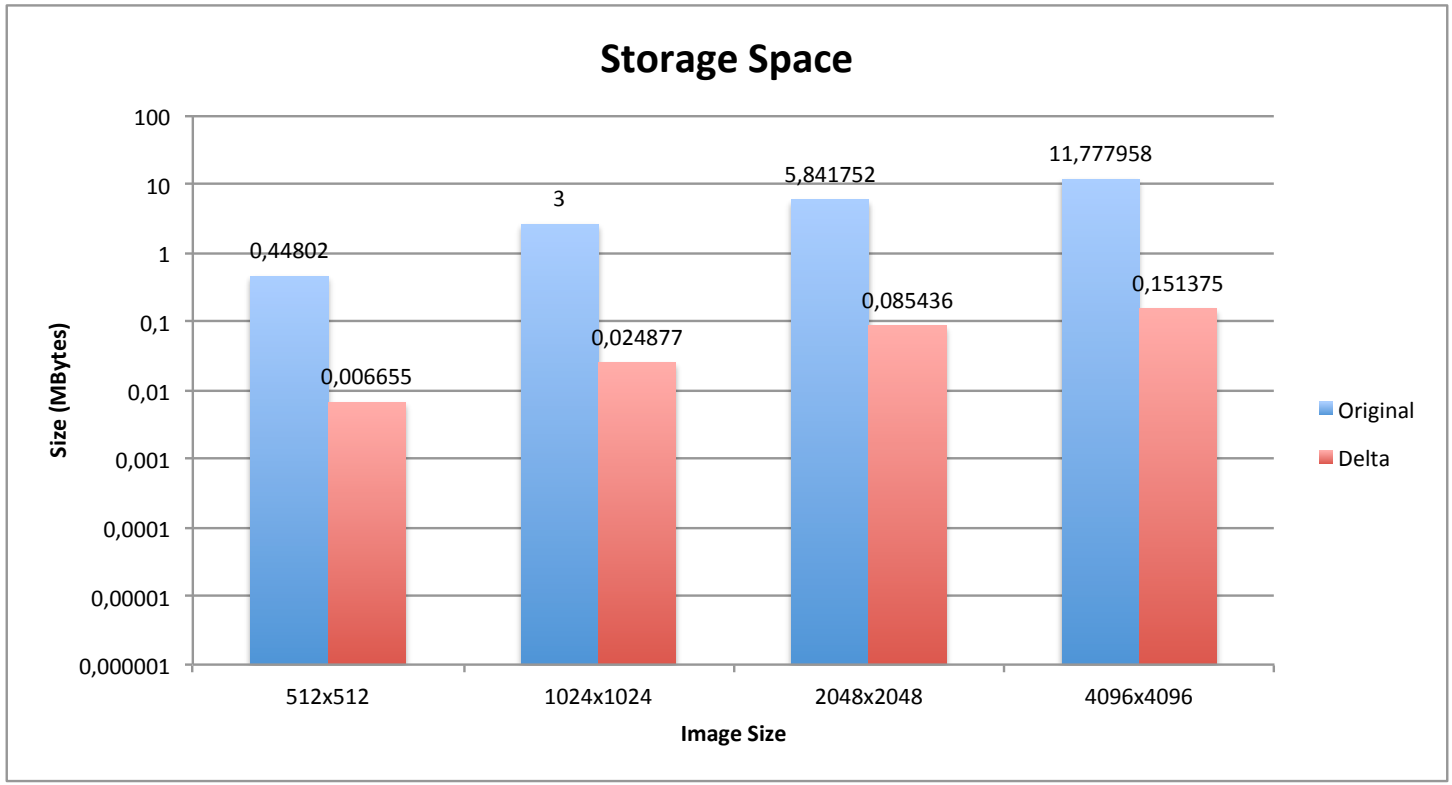
- Processing time:



*Using log_{10} scale.

Results

- Comparison between storage for the whole binary data and our delta.



*Using the "Where is Waldo" sample.

*Using log_{10} scale.

Restrictions

- Aligned images.
  - Reasonable for VCS as its track evolutions.
- All evolutions must maintain the same image's resolution.
- Only work for PNG images.

Future Works

- IMUFF is not a VCS as it.
  - Instead, provides infrastructure to allow any VCS to better work with images artifact.
- We are planning to develop a plugin for Git and Subversion to deal with image artifacts.
- Study how to work with movie artifacts.

Conclusion

- Using GPU for VCS processing can speedup up to 55x CPU processing.
- Working with delta for image artifacts can reduce greatly the space required to store it.
- Using IMUFF gives the user sematic over its image artifacts.
- Allow faster check-in and –out of image artifacts, also reducing network bandwidth for distributed VCS systems.

A GPU-based Architecture for Parallel Image-aware Version Control

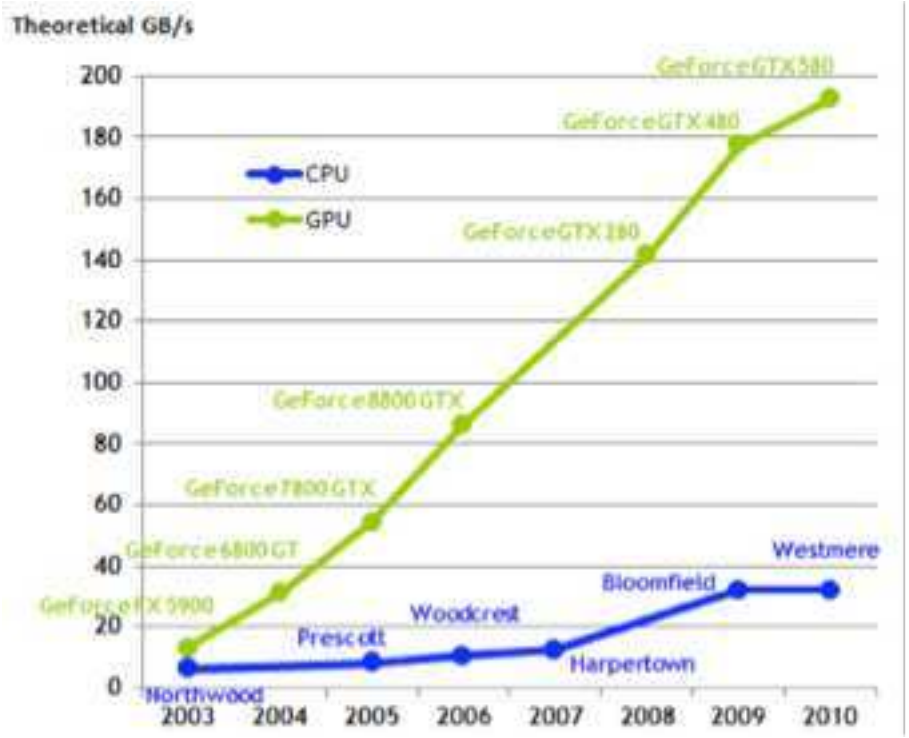
Jose Ricardo da Silva Junior Toni Pacheco
Esteban Clua Leonardo Murta

Instituto de Computação

Universidade Federal Fluminense

Using GPU for VCS operations

- GPU (Graphics Processor Unit) is a massively multi-threaded processor capable of perform almost thousands of operations/second.



- Presented in almost every personal computer!

Using GPU for VCS operations

- Allows for heterogeneous environment.
  - Both GPU and CPU doing different tasks at the same time.