Parallel Processing of Images in Mobile Devices using BOINC

David Calle, Alfredo Santamaría, David Suarez, Leonardo Florez-Valencia, and Mariela Curiel
Pontificia Universidad Javeriana de Bogotá, Departamento de Ing. de Sistemas, Bogotá, Colombia
Overview

• Research Question
• The Problem
• Challenges:
  • The Wrapper for Android Mobile Devices
  • Cross-compilation of ITK Modules
  • Parallel Processing
  • Preliminary Performance Evaluation
• Conclusions
Research Question

• It is possible to do parallel processing of images with mobile devices?
Why in Mobile Devices?

• The current capabilities of mobile devices have increased considerably.
• Nowadays mobile devices are more powerful. They have powerful processors, wide memories, large screens and open operating systems that encourage application development.
Why on Mobile Devices?

- They can perform computationally intensive operations enabling these devices to be considered as computing platforms.
- They are designed to be taken everywhere with us.
- It is necessary to develop the technology that allows us to migrate part of the problems that are solved in parallel architectures.
Why on a Grid?

- Communicate mobile devices using MPI
- Use a Grid
Why on a Grid?

- Many research projects have addressed the problem of incorporating mobile devices to the grid.

- New categories of grids have emerged: Mobile grids. Furthmüller and Waldhorst (2010) define a mobile grid as a grid that includes at least a mobile device.

- Grid technology offers added values, for example: resource discovery, scheduling of tasks, redundancy, quality of service.
Why is BOINC chosen?

• In a previous work was evaluated the existing technology for Mobile Grid

• Although many articles describing middleware and tools have been written, not all of these systems are available for use. In other cases they do not have good documentation or offer adequate support, because they are not active projects.
Why is BOINC chosen?

• Boinc provides code to run applications on mobile devices.
• It is possible to run tasks, written in various programming languages, in several execution platforms and operating systems. In particular C++ code.
• It has good documentation.
• Boinc is an active project with many support.
The Problem

• In this first step of the research we selected filtering algorithms because of simplicity and because they can be used to process different regions of one image independently.

• We use a filter that only needs the information of some neighboring pixels: the SmoothingRecursiveGaussian ImageFilter of ITK [13].
This filter generates a smooth image by performing a convolution on it. This convolution uses a kernel with values given by the Gaussian distribution.
Challenges

1) **Transparency for the programmer**: the execution of jobs in the mobile grid should not require changes in the source code.

2) **Cross-compilation** of the libraries for image processing (i.e. ITK).

3) **Distribution and collection** of data for parallel processing.

4) **Achieve adequate performance**.
First Challenge: Execution without modifying the source program

• For a task to be executed on a BOINC platform, you must modify the programs to include calls to the BOINC API or use available wrappers for the executing platform.

• At the beginning of this work, a compiled wrapper for Android operating system was not available in BOINC distribution.
First Challenge: Execution without modifying the source program

- The first task: **compile the wrapper for Android Platform**
- Current Android OS phones run on ARM CPUs but they do not have a pre-installed native compiler.
- Hence, **tools and application code for ARM need to be generated on a platform where a C++ compiler is available.**
Crosscompile

Wrapper

Library for Image Processing (ITK)

Program

BOINC:FAST 2017
Modify wrapper.cpp

Requirements for compilation the wrapper
- Android Toolchain
- Curl
- Open SSL

Requirements for execution
- The wrapper should be a PIE
  - `-fPIE -pie`
- The call of PWD was changed by the function `getcwd`

$ ./build_curl_arm.sh
$ ./build_openSSL_arm.sh
$ ./build_wrapper_arm.sh
Cross-compilation of the library for image processing (i.e. ITK [13]).

• ITK is a library for performing registration and segmentation of images.

• ITK together with VTK [25], KWWidgets [26], and IGSTK [27] are the open-source most used libraries in the field of processing and analyzing medical images.

• In order to use this library together with BOINC's infrastructure it is necessary to generate the executables for the target platform, with the help of the CMake tool.
Cross-compilation of the libraries for image processing (ITK).

• It was possible to crosscompile the modules used in our problem: Core, Filtering and Smoothing Recursive Gaussian Filter.

• It was necessary to deactivate the test flag BUILD TESTING:
  When this flag is ON, some tests are made at the end of library construction to check the product. However, these tests assume that the system is running in the target platform.
Cross-compilation of the libraries for image processing (i.e. ITK [13]).

• By default, the resulting executable code that uses the ITK library, loads libraries at runtime.
• It may not be easy to ensure that the target mobile devices have these libraries.
Cross-compilation of the libraries for image processing (i.e. ITK [13]).

**Solution:**

- It was necessary to disable these options by turning the flags ITK DYNAMIC-LOADING and BUILD SHARED LIBS to the 0 value. This ensures that the generated executable is static without any shared dependencies.

- This solution increases the size of the executable.
Distribution and collection of data for parallel processing.

The next step is to use BOINC for the parallel processing of the image. It is necessary:

a) Divide the image and to allocate pieces to each workunit.

b) Collect the outputs to generate the final result.

Distribution and collection of data for parallel processing.

The Work Generator creates the workunits by associating programs and parameters.

create_work is the command-line tool for submitting jobs

```bash
create_work [ arguments ] infile_1 ... infile_n
```
Division of the Image and generation of workunits

Work Divisor
Divides the image and put the sections (N) in different files (N files)

Adapter
This script uses `create_work` to create N workunits with the corresponding files.

N and the files
Save the job identifiers

Create_Work
(Boinc)

WorkGenerator

BOINC:FAST 2017
Distribution and collection of data for parallel processing

Assimilator

• By default, once a job is validated and assimilated, BOINC deletes all the files uploaded by the client. Hence the need to immediately process the results or save them.

• The objective of the Assimilator is to operate over the individual results generated by each finished workunit to merge the collected data.
Distribution and collection of data for parallel processing

• Two classes were implemented: the *ConcreteObserver* and *Observed*

• The code of the *ConcreteObserver* detects when a workunit is ready to be assimilated, and notifies the changes to the *Observed* class.

• Algorithms of the *Observed* class join the image.
Experimental Design

2k Factorial Design

• This experimental design pretends to determine the impact of $k$ factors on a response variable, each of which has 2 alternatives or levels.
Base Test

• We execute the *Smoothing Recursive Gaussian ImageFilter* on a single desktop computer and without the use of BOINC.

An 8 core Intel Core i7-920XM 2.0 GHz processor, 4Gb of RAM and running Linux x86 64 Ubuntu 14.04 operating system.
Base Test

- tiff image of 301 Mb, 30576 pixels wide and 9860 pixels high.
- The image selected was taken from the LROC project (The Lunar Reconnaissance Orbiter Camera) [29].

The running time of the base test was 49 minutes.
Factors

• **Processing Devices (mobile or desktop):** This factor refers to technical characteristics of BOINC clients that will process the image.

• **Degree of parallelism:** This factor indicates the number of regions in which the original image is divided and, as a result, the number of workunits required to complete the total processing of the image.
Factors

• **Redundancy**: This factor specifies how many results are required for a workunit to be successfully completed.

• **Server Location**: This factor is to determine the impact of the latency due to server's location.

  **A local server**: The first one was a computer with Ubuntu 14.04, 4Gb of RAM, 8 CPUs and 50Gb of hard disk, located in the same network as the processing devices;

  **A remote server**: the second one was a VPS provided by AWS (Amazon Web Services) with Ubuntu 14.04, 1Gb of RAM, 1 VCPU and 10Gb of SSD, located in a remote network.
Factors and Levels

Table 2. Factors and Levels

<table>
<thead>
<tr>
<th>Factor</th>
<th>Low Level (-1)</th>
<th>High Level (+1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Devices, A</td>
<td>Linux x86_64 computers</td>
<td>Mobile devices Android ARM7</td>
</tr>
<tr>
<td>Degree of Parallelism, B</td>
<td>450</td>
<td>1000</td>
</tr>
<tr>
<td>Redundancy, C</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Server, D</td>
<td>Local</td>
<td>Remote</td>
</tr>
</tbody>
</table>
## Devices Characteristics

### Table 1. Technical Characteristics of the Devices

<table>
<thead>
<tr>
<th>Number of Devices</th>
<th>Architecture</th>
<th>Model</th>
<th>CPU</th>
<th>Cores</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Linux x86_64</td>
<td>Lenovo thinkPad W510</td>
<td>Intel Core i7-920XM 2.0 GHZ</td>
<td>8</td>
<td>8Gb</td>
</tr>
<tr>
<td>5</td>
<td>Linux x86_64</td>
<td>Lenovo ThinkCentre Serie M</td>
<td>Intel Core i7 6700T 2.8 GHZ</td>
<td>8</td>
<td>8Gb</td>
</tr>
<tr>
<td>4</td>
<td>Android ARM7</td>
<td>Samsung Galaxy S4</td>
<td>Quad-core 1.9 GHz Krait 300</td>
<td>4</td>
<td>2Gb</td>
</tr>
<tr>
<td>1</td>
<td>Android ARM7</td>
<td>Nexus 4</td>
<td>Quad-core 1.5 GHz Krait</td>
<td>4</td>
<td>2Gb</td>
</tr>
</tbody>
</table>
Results

Table 3. Results of the $2^k$ experiment

<table>
<thead>
<tr>
<th></th>
<th>desktop computers</th>
<th>mobile devices</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>450 workunits</td>
<td>1000 workunits</td>
</tr>
<tr>
<td>Local</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Redundancy 1</td>
<td>15min</td>
<td>37min</td>
</tr>
<tr>
<td>Redundancy 2</td>
<td>38min</td>
<td>1h22min</td>
</tr>
<tr>
<td>Remote</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Redundancy 1</td>
<td>48min</td>
<td>59min</td>
</tr>
<tr>
<td>Redundancy 2</td>
<td>1h37</td>
<td>1h54min</td>
</tr>
</tbody>
</table>

Table 4. Percentage per factor

<table>
<thead>
<tr>
<th></th>
<th>A (Devices)</th>
<th>B (Parallelism)</th>
<th>C (Redundancy)</th>
<th>D (Server)</th>
<th>AB</th>
<th>AC</th>
<th>AD</th>
<th>BC</th>
<th>BD</th>
<th>CD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0%</td>
<td>6%</td>
<td>47%</td>
<td>30%</td>
<td>2%</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
<td>0%</td>
<td>6%</td>
</tr>
</tbody>
</table>
Results

Local vs Remote server

- Linux-Local Server
- Android- Local Server
- Linux-Remote Server
- Linux (Base test)
- Android Remote Server

Time (min)
Results
Results
Conclusions

• It was possible to compile the wrapper of BOINC for Android mobile devices.

• Part of ITK library was compiled for Android. We generate a static executable including the Core and Filtering ITK modules.
Conclusions

• BOINC supports the distribution of tasks in the computing platforms. However, it was necessary to provide additional programs to divide the data and collect the results.

• With regard to performance:
  - Execution times of images processing were very similar using mobile devices and desktop computers.
  - We recommend to place BOINC in a local Server. No redundancy
  - BOINC system is not intrusive.