9/9/2013

Haifa university

LibTune Tool

BSc graduation project

Authors: Ofir Cohen, Dror Fadida  
 (300816881, 039062658)  
Supervised by: Hananel Hazan,  
 Dr Gadi Haber

Last revised: 09/09/2013

Table of Contents

[Project draft 1](#_Toc365649102)

[Project overview 3](#_Toc365649103)

[What is LibTune? 3](#_Toc365649104)

[What is Pin? 3](#_Toc365649105)

[Client requirements 4](#_Toc365649106)

[Main requirements 4](#_Toc365649107)

[Bonus requirements 4](#_Toc365649108)

[Work plan 5](#_Toc365649109)

[Processes & main flow of the tool 6](#_Toc365649110)

[Pin education 6](#_Toc365649111)

[PinTools 6](#_Toc365649112)

[Replacing a Routine in Probe Mode 6](#_Toc365649113)

[LibTune flow 6](#_Toc365649114)

[Initialization time 7](#_Toc365649115)

[Instrumentation time 7](#_Toc365649116)

[Run-time 8](#_Toc365649117)

[Algorithm and main flow 9](#_Toc365649118)

[User manual 10](#_Toc365649119)

[Overview 10](#_Toc365649120)

[Requirements 10](#_Toc365649121)

[How to install LibTune 10](#_Toc365649122)

[How to run 10](#_Toc365649123)

[Tool knobs 11](#_Toc365649124)

[How to measure execution time? 12](#_Toc365649125)

[Developer’s manual 14](#_Toc365649126)

[What’s in the package 14](#_Toc365649127)

[Environment 14](#_Toc365649128)

[Build 14](#_Toc365649129)

[Code structure 15](#_Toc365649130)

[Software testing (QA) 16](#_Toc365649131)

[Technical difficulties 21](#_Toc365649132)

[Binary instrumentation (Pin) 21](#_Toc365649133)

[Porting project to Windows 21](#_Toc365649134)

[Compiler research 21](#_Toc365649135)

[Peculiarities of Windows 22](#_Toc365649136)

[Conclusions 23](#_Toc365649137)

[Math optimizations 23](#_Toc365649138)

[Memory optimizations 23](#_Toc365649139)

[General optimizations 23](#_Toc365649140)

[Vectorized string optimizations 24](#_Toc365649141)

[Final conclusion 24](#_Toc365649142)

[Release notes & disclaimers 25](#_Toc365649143)

[Literature and references 27](#_Toc365649144)

[Appendix 28](#_Toc365649145)

[Monthly reports (including meeting summaries) 28](#_Toc365649146)

[Notes & tasks 40](#_Toc365649147)

[High-level plan 40](#_Toc365649148)

[Low-level plan 40](#_Toc365649149)

[Software Requirements Specification 45](#_Toc365649150)

[Introduction 45](#_Toc365649151)

[General Description 46](#_Toc365649152)

[Specific Requirements 48](#_Toc365649153)

[Non-Functional Requirements 55](#_Toc365649154)

# Project draft

\* Note: this draft was proposed at the beginning of the year

|  |  |  |
| --- | --- | --- |
| שם סטודנט א :דרור פדידה | תז:039062658 | |
| שם סטודנט ב : אופיר כהן | תז:300816881 | |
| שם הפרויקט | libtune | |
|  |  | |
| מנחה:דר' גדי הבר |  | |
| טלפון: | נייד:054-4588668 | Email:gadi.haber@intel.com |
|  |  | |
|  |  | |
| מטרת הפרויקט: | הרחבה ופרודוקטיבציה של כלי אוטומטי לשיפור ביצועים של קבצי ריצה ע"י החלפה של פונקציות ספריה קיימות בפונקציות אופטימליות על גבי פלטפורמת windows | |
| דרישות קדם: | 1. קורס מבנה מהדרים 2. מבוא למערכות הפעלה 3. ידע בשפות תכנות : C/C++ | |
| שלבי ביצוע תוכנית עבודה | 1. שלב הpackging - יצירת גרסת download 2. שלב ההרחבה – החלפה של פונקציות נוספות מ lib pthread ,טיפול בפונקציות סטטיות והוספת הדפסות בקרה. 3. מדידות של ביצועים וdebug על תכניות benchmark. | |
| שלבים עיקריים ומטרות ראשיות | 1. לימוד והכרה של pin של libtune ושל ספריות pthread ו C. 2. מימוש גרסת download נוחה לשימוש. 3. מימוש של אופטימיזציות והחלפות לפונקציות ספריה נוספות. 4. מדידת ביצועים | |
| שלבי בונוסים | 1. הוספת אופטימיזציות חדשות. 2. טיפול בספריות סטטיות. 3. שיפור ביצועים של תכניות ספסל החל מ 20% ומעלה. | |
| סביבה נידרשת | Windows, Intel pin | |
| קו סיום משוער | 9.9.13 | |
| ספרות ומאמרים | 1. Intel pin manual 2. Libtune project documentation 3. libC +libpthread manuals | |

# Project overview

## What is LibTune?

LibTune is a Dynamic Binary Instrumentation tool that allows the user to optimize an already compiled application, without changing the source code. That is, the optimizations offered by this tool are related to load-time and link-time optimizations. LibTune is actually a Pin tool.

## What is Pin?

Pin is a dynamic binary instrumentation framework for the IA-32 and x86-64 instruction-set architectures that enables the creation of dynamic program analysis tools.

The tools created using Pin, called Pin tools, can be used to perform program analysis on user applications in Linux and Windows.

This project was focused on Windows platform implementation only. As a dynamic binary instrumentation tool, instrumentation is performed at run time on the compiled binary files. Thus, it requires no recompiling of source code and can support instrumenting programs that dynamically generate code.

LibTune was designed to optimize common memory and trigonometric library functions. A special test program was developed and added with the LibTune code which allows a fast analysis of the potential improvement.

# Client requirements

## Main requirements

* Port the product from Linux to Windows

Description: the product was initially designed for Linux only.

Most of the product work was to port it to Windows, design and implement a common interface for both OSs.

* Packaging & productization of the tool

Description: we provided an easy-to-use download & documents that describes how to install the tool, and use its various knobs

* Dump a neat report that summarizes which functions were replaced

Description: we generate a report that details for each image (binary/dll) which functions were replaced.

\* Note: the “tick symbol” means that these were successfully accomplished

## Bonus requirements

* Adding new optimizations

Status: completed, added memcpy, memset, malloc & free optimizations

* Handling applications with functions from static libraries

Status: completed, looking for functions in main executable too (in addition to dynamic libraries)

# Work plan

* Education about the Pin tool, Pin virtual machine architecture, OS concepts
* Ramping up on Pin by building a simple Pin tool (“memory leak” tool)
* Performing research on Intel Compiler and Microsoft Windows static & shared libraries

Understanding the different images provided by each compiler, and finding which functions exist in which image

* Ramping up on LibTune’s existing code and design
* Designing the common interface to bridge the gap between Windows & Linux
* Adding new instrumentations to LibTune
* Validation correctness (QA)
* Performance assessement
* Code cleanup and documentation
* Packaging of the tool
* Providing sample applications for the benchmark team
* External website for the Tool with links to resources

# Processes & main flow of the tool

The process of the tool is a 3-fold stages which will be detailed in the following sub-sections. First we must understand the basic flow of Pin.

## Pin education

It is important to note that LibTune works in “Probe mode” only, and **not** in “JIT mode” (we will shortly explain).

Pin intercepts the execution of the first instruction of the executable and generates ("compiles") new code for the straight line code sequence starting at this instruction. It then transfers control to the generated sequence. The generated code sequence is almost identical to the original one, but Pin ensures that it regains control when a branch exits the sequence. After regaining control, Pin generates more code for the branch target and continues execution. Pin makes this efficient by keeping all of the generated code in memory so it can be reused and directly branching from one sequence to another.

### PinTools

Conceptually, instrumentation consists of two components:

* A mechanism that decides where and what code is inserted
* The code to execute at insertion points

These two components are instrumentation and analysis code. Both components live in a single executable, a *Pintool*. Pintools can be thought of as plugins that can modify the code generation process inside Pin.

### Replacing a Routine in Probe Mode

Probe mode is a method of using Pin to insert probes at the start of specified routines. A probe is a jump instruction that is placed at the start of the specified routine. The probe redirects the flow of control to the replacement function. Before the probe is inserted, the first few instructions of the specified routine are relocated. It is not uncommon for the replacement function to call the replaced routine.

In probe mode, the application and the replacement routine are run natively. This improves performance, but it puts more responsibility on the tool writer. Probes can only be placed on RTN boundaries.

## LibTune flow

The tool flow is comprised of three main stages. In this section we will thoroughly explore each of these stages using diagrams accompanied by textual descriptions.

### Initialization time

Based on the user’s choice (the tool knobs he is using with the tool), LibTune registers the corresponding callback functions, i.e. functions that will be invoked when the images (DLL/EXE files) of the functions to be replaced are loaded.

#### Data flow diagram



Figure - demonstrates the interaction of user and LibTune

### Instrumentation time

When callbacks are invoked we search for the pre-defined set of functions (involved with each particular callback) in the image, and if we find the functions we are looking for, and the replacement is considered valid (see RTN\_IsValid() and RTN\_IsSafeForProbedReplacement() Pin API calls documentation in the Pin manual website for more information), we apply the replacement with the optimized version of these functions.

#### Sequence diagram

Figure - demonstrates how (application) functions are replaced with their optimized versions

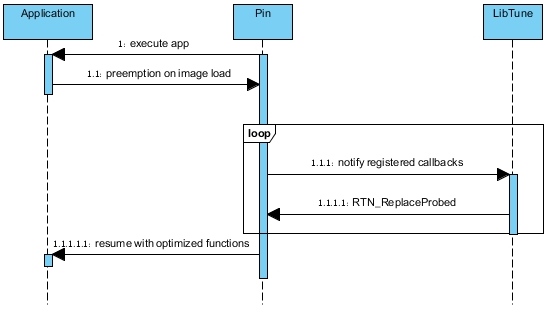


Figure - demonstrates how (application) functions are replaced with their optimized versions

### Run-time

After LibTune callback functions were invoked (on image load event), at run-time we could load either of two replacement functions:

1. Wrapper function (indirect jump to replacement function) –

There are some cases where we don’t necessarily want to invoke the optimized versions of the target functions, and a decision strategy has to be taken at run-time.

Such cases include **parallel** optimizations, for instance **–parallel\_memcpy 4**

Will check (at run-time) if it is worthy to invoke the parallel implementation, worthy in this case means if the buffer size is bytes, or if the set of threads are already working on another **memcpy** task and we must wait for the threads to finish.

Therefore, if either of the cases above was true, we invoke the non-parallel version of the function.

Another case where we need wrapper function is when parameters of functions we want to replace have different type or order (in the function signature), and therefore we have to cast/re-order the arguments before invoking the original function.

The last case would be the parallel optimizations, where we need to prepare (slice) the work to the available threads prior to invoking them.

1. Direct jump (to replacement function)

This is the case where functions being replaced have identical signature to the replaced function, and in most cases it isn’t necessary to determine at runtime the worthiness (for example if the optimized version always performs better).

## Algorithm and main flow

In this project, LibTune is running in “Probe mode” and not “JIT mode” (see reference to “Pin manual” on the references section for more information).

When working in “Probe mode” Pin insert probes (jump instructions) at the beginning of the routines in question. As we mentioned, probe is actually a jump instruction that is placed at the beginning of the specified routine.

The probe changes the original flow of control of the application to actually jump to the replacement routine instead of the original one. The replacement function is able to invoke the replaced (original) routine (e.g. see the wrappers mentioned in the previous sub-section).

When In probe mode, the application and the replacement routine run natively. This improves performance but implicates that the tool writer be more cautious with regard to the code of the tool.

The flow of an application running under probe mode is as follows:

1. Pin is loaded
2. Pin loads the target application
3. At “image load” callback event, Pin invokes all the subscribed callbacks and they decide

which functions to instrument, and do the replacement via Pin.

1. After target application is patched (functions are replaced), Pin starts the application and now the old functions (target functions) will not be invoked, instead only the optimized (replacement) functions are invoked at run-time.

For specific information about the inner details of the algorithm of LibTune, that is information about its knobs and how it operates, please refer to the “Developers Manual” section in this document.

# User manual

## Overview

**LibTune** is a Dynamic binary instrumentation tool that allows the user to optimize an already compiled application, without changing the source code. LibTune is actually a Pin tool.

**What is Pin?** Pin is a dynamic binary instrumentation framework for the IA-32 and x86-64 instruction-set architectures that enables the creation of dynamic program analysis tools. The tools created using Pin, called **Pin tools**, can be used to perform program analysis on user applications in Linux and Windows. This project was focused on Windows platform implementation only. As a dynamic binary instrumentation tool, instrumentation is performed at run time on the compiled binary files. Thus, it requires no recompiling of source code and can support instrumenting programs that dynamically generate code. LibTune was designed to optimize common memory and trigonometric library functions. A special test program was developed and added with the LibTune code which allows a fast analysis of the potential improvement.

## Requirements

* **CPU** - All IA64 and Intel64 processors. An IA32 processor needs the following features:

C8 - CMPXCHG8B

SSE2

FXSR - FXSAVE/FXRSTOR

* **Operating System -** Microsoft Windows of the following versions:

Windows XP, Windows Vista and Windows 7, 32 and 64 bit versions

Windows Server 2003, 2008 and 2008 R2

* Target applications : applications compiled with Microsoft Visual Studio Compiler

## How to install LibTune

1. Download Pin (revision 56579) from: http://software.intel.com/sites/landingpage/pintool/downloads/pin-2.12-56759-msvc10-windows.zip
2. Extract the LibTune archive to some folder, let's assume it is extracted to:   
   C:\pin\pin-2.12-56759-msvc10-windows\
3. Extract LibTune archive to:

C:\pin\pin-2.12-56759-msvc10-windows\source\tools\libtune\

## How to run

1. Start a new DOS (cmd) session, and go to:

C:\pin\pin-2.12-56759-msvc10-windows\source\tools\libtune\

1. To see the tool knobs, do the following:

..\..\..\pin -t obj-intel64\libTune.dll -toolhelp 1 -- "tested\_ofir\2. -math knob\atan\_test\_64bit.exe"

1. Execute (64-bit):
   1. set PATH=%PATH%;lib\x64
   2. Run: ..\..\..\pin -t obj-intel64\libTune.dll <tool knobs> -- <application<

For example:

..\..\..\pin -t obj-intel64\libTune.dll -math 1 -- "tested\_ofir\2. -math knob\atan\_test\_64bit.exe"

OR use the automated script in the following section (how to measure execution time(

1. Execute (32-bit):
   1. set PATH=%PATH%;lib\x86
   2. Run: ..\..\..\pin -t obj-ia32\libTune.dll <tool knobs> -- <application<

For example:

..\..\..\pin -t obj-ia32\libTune.dll -math 1 -- "tested\_ofir\2. -math knob\atan\_test\_32bit.exe"

OR use the automated script in the following section (how to measure execution time(

Note: the tool generates a log file, pintool.log, in the tool's directory:

C:\pin\pin-2.12-56759-msvc10-windows\source\tools\libtune\pintool.log

Where you can see which functions were actually replaced.

## Tool knobs

1. **Math optimizations knobs**:
2. Syntax: -math 0/1

Description: replaces various math functions (from <cmath> header file), for example: cos,sin,tan,etc.

1. Syntax: -mkl 0/1

Description: replaces various math functions (from <cmath> header file), for example: cos,sin,tan,etc

1. **Memory optimizations knobs**:
2. Syntax: -mem 0/1

Description: replaces "memcpy", "memset" functions with Intel IPP functions

1. Syntax: -malloc\_free 0/1

Description: replaces "malloc" & "free" functions with their Intel IPP functions

1. Syntax: -read\_ahead 0/1

Description: enables the read-ahead optimization (see disclaimers below)

1. **Parallel memory optimizations knobs**:
2. Syntax: -parallel\_memcpy <number of threads>

Description: replaces application's "memcpy" functions with a parallel implementation of "memcpy"

1. Syntax: -parallel\_memset <number of threads<

Description: replaces application's "memset" functions with a parallel implementation of "memset"

1. **String optimizations knobs:**
2. Syntax: -vectorized\_strlen 0/1

Description: replaces the "strlen" function with a vectorized implementation

1. Syntax: -vectorized\_strchr 0/1

Description: replaces the "strchr" function with a vectorized implementation

1. Syntax: -vectorized\_strstr 0/1

Description: replaces the "strstr" function with a vectorized implementation

1. **All optimization knob:**

Syntax: -all 0/1

Description: invokes all optimizations knobs above ( number of threads in parallel memory optimizations is 4).

Note: you can mix between (2) and (3) i.e. replaces user memory functions with parallel ones (Intel or default functions only in parallel mode(

## How to measure execution time?

As explained before, the LibTune application enables replacing standard functions with better implemented functions during program run time. This optimization can be measured in many aspects, one of them is running time (defined by the time passed from the moment program started to the moment it finished). One simple way to do so is by using the added script *TimeMeasure.bat*.

1. Start a new cmd session, and go to:

C:\pin\pin-2.12-56759-msvc10-windows\source\tools\libtune\tested\_ofir\2. -math knob\

1. Run: lib\_run64.bat, or lib\_run32.bat (depending on your choice)

Output Of Example

---------------------

Running atan test

---------------------

1 file(s) copied.

\*\*\*\*\* Pin Runtime \*\*\*\*\*

Initializing LibTune v1.1

command took 0:0:3.13 (3.13s total)

\*\*\*\*\* Pin Runtime \*\*\*\*\*

\*\*\*\*\* App Runtime \*\*\*\*\*

command took 0:0:3.43 (3.43s total)

\*\*\*\*\* App Runtime \*\*\*\*\*

Note: Pin Runtime includes Pin and Libtune "overhead"

For most extensive and versatile benchmarks, use external measurement tools like SPEC2006

# Developer’s manual

## What’s in the package

* *Supplemental libraries* (libraries that will provide the replacement functions)

These include: ICC, IPP and MKL redistributable DLLs

These can be found under **lib/<arch>** directory.

* *Build utilities****:*** makefiles and “batch files” (see *Build* below)
* *Source code*: all .cpp and .h files under the project directory
* *Sanity tests*: under tests/

## Environment

The following has to be installed prior to developing/building the code

* Platform: Windows 7/8, 32-bit or 64-bit
* Microsoft Visual Studio 2010
* Cygwin with make
* Intel Compiler Suite (see link in the references section)

## Build

The project is built using the ”make” utility of Linux (Cygwin).

There are two makefiles that determine the way of building the project:

1. makefile.rules
2. libtune.make.config

In order to build it you need to:

1. Download Pin & Extracting LibTune to the right folder

(Please refer to the “user manual” for more information)

1. Open a “Visual Studio Command Prompt” (shipped with the Visual Studio installation)
2. Go to the “libtune” folder (under Pin’s folder), and run
3. lib\_make64.bat – if you’re building a 64-bit LibTune
4. lib\_make32.bat – if you’re building a 32-bit LibTune

These “batch files” take care of preparing the environment for ICC (Intel Compiler Collection) before launching the compiler, and launches the “make” command with the relevant parameters.

Once this command has completed, the DLL of LibTune is built under obj-intel64/obj-ia32 for 64-bit/32-bit platform.

In order to ship LibTune to users, you need to provide the redistributable DLL files and the LibTune DLL’s (for both 32-bit and 64-bit).

Before redistributing DLL files of Intel Compiler Suite (including IPP, MKL and IPP), please consult Intel’s legals department, Intel’s Compiler Suite website, read the terms and legal issues that might arise and follow Intel’s exact rules to understand what files are/aren’t to be shipped and.

## Code structure

In this section we will thoroughly explore the structure of the code in this project.

Let’s start by presenting a high-level description of the code.



Figure - the high-level description of the code structure

Now we will detail each file in the project.

File: libTune.cpp

# Software testing (QA)

Note: TC = Test Case

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| TCID | TC NAME | ACTIONS | EXPECTED | ACTUAL RESULT | |
| WIN32 | WIN64 |
|  | Show tool usage info | Running tool with “-tool\_help 1” | Prints tool usage info to console | PASSED | PASSED |
|  | Measure Pin Runtime Overhead | Running tool with no knobs | Runs the program with no instrumentation | PASSED | PASSED |
|  | Invalid knobs | Running tool with invalid knob | Print tool usage info to console | PASSED | PASSED |
|  | Math instrumentations | Running tool with “-math 1” | All pre-defined math functions will be replaced with ICC implementation | PASSED | FAILED: due to limitation (#1) of Microsoft images (see the following disclaimers section for more info) |
|  | MKL instrumentations | Running tool with “-mkl 1” | All pre-defined math functions will be replaced with MKL implementation (with performance warning) | PASSED | FAILED: due to limitation (#1) of Microsoft images (see the following disclaimers section for more info) |
|  | Memory management instrumentations | Running tool with “-mem 1” | All pre-defined memory functions will be replaced with IPP implementation | PASSED PARTIALLY: due to limitation (#5) | PASSED PARTIALLY: due to limitation (#5) |
|  | Memory allocation instrumentations | Running tool with “-malloc\_free 1” | malloc and free functions will be replaced with IPP implementation | PASSED | PASSED |
|  | Read ahead (prefetch) instrumentations | Running tool with “-read\_ahead 1” | read() and close() low-level I/O functions will be replaced by custom implementation | PASSED PARTIALLY: due to limitation (#7) | PASSED PARTIALLY: due to limitation (#7) |
|  | Memcpy Parallel instrumentations  (with buffer size less than 300,000 bytes) | Running tool with  “-parallel\_memcpy <#threads>” | Memcpy ***WILL*** not be replaced by a custom multi-thread implementation | PASSED | PASSED |
|  | Memcpy Parallel instrumentations (with buffer size > 300,000) | Running tool with  “-parallel\_memcpy <#threads>” | Memcpy ***WILL NOT*** be replaced by a custom multi-thread implementation | PASSED | PASSED |
|  | Memset Parallel instrumentations  (with buffer size less than 300,000 bytes) | Running tool with  “-parallel\_memset <#threads>” | Memset ***WILL*** not be replaced by a custom multi-thread implementation | PASSED | PASSED |
|  | Memset Parallel instrumentations (with buffer size > 300,000) | Running tool with  “-parallel\_memset <#threads>” | Memset ***WILL NOT*** be replaced by a custom multi-thread implementation | PASSED | PASSED |
|  | Unsupported knobs for Windows | Running tool with “-parallel\_memchr <#threads>” | Error message will be printed to console for unsupported knob on Windows | PASSED | PASSED |
| Running tool with “-parallel\_memcmp <#threads>” | Error message will be printed to console for unsupported knob on Windows | PASSED | PASSED |
| Running tool with “-parallel\_memmem <#threads>” | Error message will be printed to console for unsupported knob on Windows | PASSED | PASSED |
| Running tool with “-parallel\_bzero <#threads>” | Error message will be printed to console for unsupported knob on Windows | PASSED | PASSED |
| Running tool with “-spinLock 1” | Error message will be printed to console for unsupported knob on Windows | PASSED | PASSED |
|  | Vectorized string length operation | Running tool with “-vectorized\_strlen 1” | strlen() will be replaced with the custom vectorized version | PASSED | PASSED |
|  | Vectorized strchr operation | Running tool with “-vectorized\_strchr 1” | strchr() will be replaced with the custom vectorized version | PASSED | PASSED |
|  | Vectorized strstr operation | Running tool with “-vectorized\_strstr 1” | strstr() will be replaced with the custom vectorized version | PASSED | PASSED |
|  | Parallel memcpy with Intel implementation and buffer >= 300,000 | Running tool with “-mem 1 -parallel\_memcpy 4” | memcpy() will be replaced with a parallel implementation of Intel | PASSED | PASSED |
|  | Parallel memcpy with Intel implementation and buffer < 300,000 | Running tool with “-mem 1 -parallel\_memcpy 4” | memcpy() WILL NOT be replaced with a single-thread implementation of Intel | PASSED | PASSED |
|  | Parallel memset with Intel implementation and buffer >= 300,000 | Running tool with “-mem 1 -parallel\_memset 4” | memset() will be replaced with a parallel implementation of Intel | PASSED | PASSED |
|  | Parallel memset with Intel implementation and buffer < 300,000 | Running tool with “-mem 1 -parallel\_memset 4” | memset() WILL NOT be replaced with a single-thread implementation of Intel | PASSED | PASSED |

# Technical difficulties

## Binary instrumentation (Pin)

Description: as noted above, Pin is the tool on which LibTune is based, it is the underlying engine of the functions instrumentations performed on the application. Understanding how Pin works was crucial in order to develop a Pin tool (LibTune in our case) and understand it’s behavior.

Solution: we held ~ 5-6 sessions (weekly meetings) with the project technical leader, Nafta Shalev, in which we were given a detailed tutorial and introduction to the Tool, How it works, understanding OS concepts such as: OS dynamic linker, linkage of application (static vs. dynamic), DLLs search path, Pin modes (probe & JIT), compiler related optimizations, limitations, etc. To conclude, we had to learn about Pin in both theory and practice.

During our introduction to the tool we were given a ramp-up exercise that required us to implement a simple “Memory Leaks Detector” tool, that is a tool that detects memory leaks of the application being instrumented.

## Porting project to Windows

Description: the LibTune project was built and designed for Linux only. It was never built or tested on Windows platform. Our first objective was to port the implementation of LibTune to Windows and validating the correctness of the tool & comparing the behavior of the tool on Windows vs. Linux. The project was written in C++.

Solution: We had to do a lot of research on Windows to discover the following:

* Finding the corresponding Linux APIs

(e.g. \_\_sync\_val\_compare\_and\_swap on Linux, vs. InterlockedCompareExchange on Windows) and corresponding functions, finding the matching types of Windows, etc

* Devising a common interface (header files) to bridge the gap between Windows & Linux by separating the OS-specific logic to different files, and writing interfaces that will be used by both OSes. Therefore maintaining the tool’s logic with minimum OS divergence in code.

## Compiler research

Description: Intel provides a compiler with rich libraries (ICC, IPP, MKL, TBB) containing optimized versions of memory & math functions. We had to discover which library we are interested in, i.e. explore each of the libraries and determine if it’s a candidate for dynamic instrumentation (if the library provides shared libraries that Pin can use to instrument the application).

Solution: reading thoroughly the compiler suite documentation and verifying the validity of the usage of these DLLs. It also means to understand the major differences between Windows & Linux in terms of the CRT library. On Linux, the compilers (GCC & ICC) provide a shared object (libc.so, libintlc.so) with implementation of all C runtime functions, whereas in Windows there’s a higher-level layer provided by the OS (kernel32.dll, ntdll.dll), and it is not possible to directly access some of the functions provided by the C library.

Therefore, optimization such as “**read ahead**” were not so trivial as they were in Linux due to the fact that low-level I/O functions are overloaded and wrapped with higher-level services provided by Microsoft. It was crucial to understand the architecture of the OS in order to determine the limitations of the tool on Windows (see disclaimers below for more information).

## Peculiarities of Windows

Description: we had added a reporting option to the tool, that dumps to a log file a report of all the functions that were replaced in the application. Although we have noticed that on 64-bit applications, a fair amount of functions were not replaced, unlike 32-bit applications where 2x more functions were found & replaced.

Solution: after deep investigation that took place during several sessions with the Pin team (in Intel) we had discovered a limitation that is inherent to the msvcrt DLL file in 64-bit in which some functions were written to a data section and not code section, therefore Pin didn’t find the functions we were looking for, and was unable to instrument them, because Pin searches for “Code section” for functions to instrument.

# Conclusions

The initial idea was to gain on Windows the same performance achieved on Linux.

After spending long time researching and understanding the architecture of the Windows OS and based on our experiences with various applications (which where the target of the tool), we’ve found that on Windows we have gained partial success of our objective to optimize, this is due to some limitations which are elaborated below.

## Math optimizations

For starters, on Windows some headers are missing, such as <tgmath.h>, and therefore **25** math functions (which were good candidate on Linux) were excluded from being optimized.

Furthermore, due to how Microsoft compiler’s linker generates the Microsoft shared library for math functions on 64-bit platforms, further **30** math functions cannot be instrumented (see section limitations for more details).

Although we have noticed significant speedup on some applications using trigonometric functions. Therefore, these optimizations are recommended to be used by the tool, although we cannot guarantee full coverage of math libraries (due to the problems we experienced on Windows).

## Memory optimizations

For malloc and free functions we saw improvement, 1.12x speedup. This happens when certain conditions are met, such as when the application is memory bounded and malloc is used extensively.

For memset we saw ~ 1.2x (up to) speedup, and for memcpy we saw ~ 1.4x (up to) speedup.

For parallel memory operations (using the -parallel knobs) we saw significant slowdown, and we do not recommend using them, or at least do more research before using these knobs (perhaps they are good for some applications).

## General optimizations

Spinlock optimization was not tested on Windows due to the absence of pthread library on Windows, and it requires a deeper research (out of the scope of this project) to actually understand how to apply this instrumentation on Windows. Therefore this knob is disabled.

Read ahead optimization is not recommended to be used, because of the limitation (see limitations sections for more information), therefore we do not recommend to use it.

## Vectorized string optimizations

On processors with Intel® SSE technology, the vectorized string operations can benefit a lot due to a good exploitation of the vector registers in the architecture, and we definitely recommend using them.

## Final conclusion

LibTune in theory can deliver good speedup for applications by replacing standard libraries with optimized ones. But, in practice due to limitations detailed above and on the limitations section, LibTune shows different results on the Windows platform.

Therefore not every target application can reap the benefits of LibTune, and one must understand the nature of the application in order to set (correctly) the speedup expectation from the application.

# Release notes & disclaimers

The following notes refer to the Windows version of LibTune only.

1. The following list of functions will not be covered with the "-math 1" knob:

"cbrt", "cbrtf", "copysignf", "copysignl", "erfcf","erff","exp2l","fdim",

"log1p","log1pf","log1pl","lroundl", "nearbyintl","nextafterl","nexttoward",

"nexttowardf","nexttowardl","remainderl","remquol","scalblnl","scalbnl", "tgamma","tgammaf","trunc","truncl”

1. When running in 64-bit mode some of the math functions will not be replaced.

Please refer to LibTune’s log file (pintool.log) to see which functions were replaced (due to limitation in MSVCR100D.DLL image where cos,sqrt,etc are in Data section(

1. The following parallel functions will not be replaced in Windows:

* -parallel\_memcmp – no IPP version or ICC dll
* -parallel\_memchr – no IPP version or ICC dll
* -parallel\_memmem – no Windows version
* -parallel\_bzero – no bzero in Windows version

1. When using –mem 1:

i.e. replacing memory functions (memset, memcpy (We are not supporting more than 4GB size of memory operations.

1. The following functions will not be replaced in Windows:

* memcmp – no IPP version or ICC dll
* memchr – no IPP version or ICC dll
* memmem – no Windows version
* bzero – no Windows version

1. MKL optimizations cause a significant slowdown to the application, therefore it is not recommended using them.

Affected knobs: -mkl

1. Read-ahead optimization is partially working.

In Windows , unlike in Linux (where there is a single system call for read() and close() ),multiple low-level I/O read() functions exist with different parameter list. Therefore , a more thorough research and investigation is needed before instrumenting these functions.

# Literature and references

* Pin – a dynamic binary instrumentation tool: <http://www.pintool.org>
* Windows API documentation:

<http://msdn.microsoft.com/en-us/library/windows/desktop/>

* Intel Compiler Suite:

<http://software.intel.com/en-us/intel-compilers>

* Intel Integrated Performance Primitives (IPP):

<http://software.intel.com/en-us/articles/intel-integrated-performance-primitives-documentation/>

* Intel® Math Kernel Library (MKL):

<http://software.intel.com/en-us/articles/intel-mkl-kb-home/>

* Dependency Walker:

<http://www.dependencywalker.com/>

# Appendix

## Monthly reports (including meeting summaries)

נתוני הפרוייקט

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **שם הפרויקט** | אינטל -  Dynamic binary instrumentation to improve performance of running applications. | | | | |
| **שם המנחה** | גדי הבר | | | | |
| **שם הסטודנט** | אופיר כהן | ת.ז. | 300816881 | email | [ofircohenn@gmail.com](mailto:ofircohenn@gmail.com) |
| **שם הסטודנט** | דרור פדידה | ת.ז. | 039062658 | email | [dror.fadida@gmail.com](mailto:dror.fadida@gmail.com) |

**סטטוס הפרוייקט לחודש \_\_\_\_ספטמבר**

|  |  |
| --- | --- |
| תאריך | 5.09.13 |
| סטאטוס )מה עשיתי החודש...( | * בדיקת נכונות של כל הפונקציות בפרויקט * הוספה של 2 אופטימיזציות: החלפה של malloc,free   בגירסאות מאופטמות מספריית IPP   * הבנה של מגבלות המוצר בפלטפורמת Windows ותיעודן בrelease notes של הכלי * ביצוע packaging של התוכנה והפצתה לצוות שמודד benchmarks בשתי גירסאות: 32-bit, 64-bit * כתיבה והרצת טסטים חדשים שמודדים זמני ריצה * תיקוני באגים * מסמכים של הפרויקט |
| בעיות מיוחדות |  |
| שינויי לו"ז |  |
| תאריך מפגש עתידי עם המנחה | 01/09/2013 |
| תוכנית עבודה לחודש הקרוב | מסמכי הפרויקט  פגישה עם המנחה, גדי הבר, לסיכום הפרויקט |
| הערות |  |
| **סיכום פגישות עם המנחה שנעשו במהלך החודש**   * הבנה של מגבלות המוצר ב Windows ותיעודן במסמכים שיופצו איתו * קיום פגישה נוספת לסקירה סופית וסיכום הפרויקט | |

**סטטוס הפרוייקט לחודש \_\_\_\_אוגוסט**

|  |  |
| --- | --- |
| תאריך | 5.08.13 |
| סטאטוס )מה עשיתי החודש...( | * בדיקת נכונות של פונקציות * Cleanup של הקוד * הרצת טסטים * פתירת בעיות קומפלציה * תיקוני באגים |
| בעיות מיוחדות |  |
| שינויי לו"ז |  |
| תאריך מפגש עתידי עם המנחה |  |
| תוכנית עבודה לחודש הקרוב | המשך בדיקות נכונות  הרצת בדיקות (טסטים/verification)  התחלת שלב הpackaging |
| הערות |  |
| **סיכום פגישות עם המנחה שנעשו במהלך החודש** | |

**סטטוס הפרוייקט לחודש \_\_\_\_יולי**

|  |  |
| --- | --- |
| תאריך | 5.07.13 |
| סטאטוס )מה עשיתי החודש...( | * לא התקדמנו בפרוייקט עקב תקופת מבחנים |
| בעיות מיוחדות |  |
| שינויי לו"ז |  |
| תאריך מפגש עתידי עם המנחה |  |
| תוכנית עבודה לחודש הקרוב | ניקיון הקוד מקטעי קוד שלא בשימוש ותיעוד נוסף.  הרצת בדיקות (טסטים/verification)  התחלת שלב הpackaging |
| הערות |  |
| **סיכום פגישות עם המנחה שנעשו במהלך החודש** | |

**סטטוס הפרוייקט לחודש \_\_\_\_יוני**

|  |  |
| --- | --- |
| תאריך | 5.06.13 |
| סטאטוס )מה עשיתי החודש...( | * המשכנו בהתגברות על בעיות הקימפול מחודש שעבר, והצלחנו * הפרדנו את הפרוייקט לקוד עבור פלטפורמת windows ופלטפורמת LINUX בצורה יותר יסודית * יצרנו שטחי עבודה משותפים עם הצוות שעובד בפלטפורמת הלינקוס * תיעדנו את השינויים שעשינו בקוד * התחלנו בהרצת בדיקות כדי לוודא שהמעבר ל windows נעשה בהצלחה. |
| בעיות מיוחדות |  |
| שינויי לו"ז |  |
| תאריך מפגש עתידי עם המנחה |  |
| תוכנית עבודה לחודש הקרוב | ניקיון הקוד מקטעי קוד שלא בשימוש ותיעוד נוסף.  הרצת בדיקות (טסטים/verification)  התחלת שלב הpackaging |
| הערות |  |
| **סיכום פגישות עם המנחה שנעשו במהלך החודש** | |

**סטטוס הפרוייקט לחודש \_\_\_\_מאי**

|  |  |
| --- | --- |
| תאריך | 5.05.13 |
| סטאטוס )מה עשיתי החודש...( | * הצלחנו להתגבר על בעית הקימפול מחודש שעבר * הפרדנו את הפרוייקט לקוד עבור פלטפורמת windows ופלטפורמת LINUX * יצרנו שטחי עבודה משותפים עם הצוות שעובד בפלטפורמת הלינקוס * תיעדנו את השינויים שעשינו בקוד * התחלנו בהרצת בדיקות כדי לוודא שהמעבר ל windows נעשה בהצלחה. |
| בעיות מיוחדות |  |
| שינויי לו"ז |  |
| תאריך מפגש עתידי עם המנחה |  |
| תוכנית עבודה לחודש הקרוב | ניקיון הקוד מקטעי קוד שלא בשימוש ותיעוד נוסף.  הרצת בדיקות  התחלת שלב הpackaging |
| הערות |  |
| **סיכום פגישות עם המנחה שנעשו במהלך החודש** | |

**סטטוס הפרוייקט לחודש \_\_\_\_אפריל**

|  |  |
| --- | --- |
| תאריך | 5.04.13 |
| סטאטוס )מה עשיתי החודש...( | * המשכנו בהמרת הפרוייקט הקיים מפלטפורמת לינוקס לווינדוס * בנינו קובץ MAKEFILE ביחד עם המנחה * תיעדנו את הישנויים שעשינו בקוד לטובת מעבר פלטפורמה כדי שהמעבר יהיה מסודר ומובנה. * תיעדנו את כל המשימות שנדרשו מאיתנו מתחילת הפרוייקט . |
| בעיות מיוחדות | כרגע הקוד לא מתקמפל , לערכתנו הבעיה נמצאת בגרסת הwindows של ה PIN , הכלי של אינטל בו אנחנו משתמשים, מחכים לתשובה מנפתלי |
| שינויי לו"ז | חידדנו את שלבי העבודה שנעשו ואת השלבים הבאים:   1. מחקר – בוצע 2. הקמת פרויקט בסביבת windows – בשלבים סופיים 3. Packaging - טרם בוצע 4. מימוש ויישום המחקר – טרם בוצע |
| תאריך מפגש עתידי עם המנחה | ככל הנראה בשבוע הבא |
| תוכנית עבודה לחודש הקרוב | המשך העבודה עפ"י הלו"ז |
| הערות |  |
| **סיכום פגישות עם המנחה שנעשו במהלך החודש**  בנינו makefile חידדנו את השלבים הבאים בפרוייקט. | |

**סטטוס הפרוייקט לחודש \_\_\_\_מרץ**

|  |  |
| --- | --- |
| תאריך | 5.03.13 |
| סטאטוס )מה עשיתי החודש...( | * עקב תקופת מבחנים לא ביצענו שום דבר חדש בפרוייקט החודש |
| בעיות מיוחדות | אין |
| שינויי לו"ז | חידוש הפעילות עוד השבוע |
| תאריך מפגש עתידי עם המנחה | ככל הנראה בשבוע הבא (המנחה במילואים) |
| תוכנית עבודה לחודש הקרוב | רענון קצר של החומרים שכבר יש לנו והמשך פגישות עם נפתלי |
| הערות |  |
| **סיכום פגישות עם המנחה שנעשו במהלך החודש**  לא היו.. | |

**סטטוס הפרוייקט לחודש \_\_\_\_פברואר**

|  |  |
| --- | --- |
| תאריך | 5.02.13 |
| סטאטוס )מה עשיתי החודש...( | * המשכנו ללמוד את הקוד של הפרוייקט LIBTUNE במטרה להפעיל אותו על מערכת ההפעלה windows (נכתב במקור לlinux). * הכנו רשימת פעולות "תכנית עבודה" ביחד עם נפתלי כדי להביא את הקוד לפעול ב windows . * התחלנו לעשות את המעבר , אנחנו בערך במחצית הדרך. * לקראת סוף החודש הפסקנו לעבוד כדי להתכונן לתקופת הבחינות. |
| בעיות מיוחדות | בעיה בבנית MAKE, הוחלט שנעשה זאת יחד עם יפתח |
| שינויי לו"ז | משהים את הפרוייקט עד לחודש הבא עקב בחינות. |
| תאריך מפגש עתידי עם המנחה | אין. |
| תוכנית עבודה לחודש הקרוב | לא נעבוד על הפרוייקט החודש |
| הערות |  |
| **סיכום פגישות עם המנחה שנעשו במהלך החודש**  לא היו.. | |

**סטטוס הפרוייקט לחודש \_\_\_\_ינואר**

|  |  |
| --- | --- |
| תאריך | 5.01.13 |
| סטאטוס )מה עשיתי החודש...( | * ראינו דוגמאות למימוש כלים שונים בעזרת הPIN * ביצענו תרגילי היכרות בתמיכת נפתלי – מימשנו כלי לזיהוי זליגות זיכרון בתוכניות "צד שלישי" הרצנו את הכלי על קבצי הbinary שלהן * עמדו על הביצועים של הכלי – מה התקורה בהרצת הכלי על התוכניות לעומת הרצה של התוכניות באופן עצמאי. * ביצענו שיפורים בקוד ושינינו את מבנה הנתונים שבחרנו כדי לשפר את זמן הריצה בעזרת דגשים שקיבלנו מנפתלי. * סיימנו את שלב ההיכרות עם PIN * התחלנו ללמוד את הקוד של הפרוייקט LIBTUNE במטרה להמשיך לפתחו |
| בעיות מיוחדות | לא היו.. |
| שינויי לו"ז | הוחלט ,שבניגוד למה שנכתב בהגדרת הפרוייקט נתחיל קודם כל בהרחבתו – הוספת פונקציות להחלפה, ושיפור הקוד הקיים וקיום מדידות בהמשך ורק אח"כ נעבור לשלב הpackaging. |
| תאריך מפגש עתידי עם המנחה | כרגע אנו נפגשים באופן קבוע אחת לשבוע עם נפתלי , כאשר גדי מעביר את הדרישות והדגשים דרכו. |
| תוכנית עבודה לחודש הקרוב | פגישות שבועיות עם נפתלי - סיום למידת הפרויקט עד לנקודה בו נעצר פיתוחו והמשך הפיתוח על ידנו. |
| הערות |  |
| **סיכום פגישות עם המנחה שנעשו במהלך החודש**  החודש לא נפגשנו עם גדי . | |

**סטטוס הפרוייקט לחודש \_\_\_\_דצמבר\_\_**

|  |  |
| --- | --- |
| תאריך | 5.12.12 |
| סטאטוס )מה עשיתי החודש...( | * קיבלנו אישור מהמנחה שנבחרנו לבצע את הפרוייקט תחתיו * פגישת פתיחה עם המנחה * יצרנו מסמך הגדרת פרוייקט * הגשנו את הגדרת הפרוייקט ואת שאר המסמכים הדרושים * נפגשנו עם נפתלי שלו – מלווה את הצד הטכני של הפרוייקט * התחלנו ללמוד את המערכת וסביבת העבודה בה נעבוד |
| בעיות מיוחדות | לא היו.. |
| שינויי לו"ז |  |
| תאריך מפגש עתידי עם המנחה | אין עדין |
| תוכנית עבודה לחודש הקרוב | פגישות שבועיות עם נפתלי – בתחילה פגישות tutorials ובהמשך תמיכה ועזרה בבעיות טכניות שניתקל בהן |
| הערות |  |
| **סיכום פגישות עם המנחה שנעשו במהלך החודש**  נפגשנו עם גדי ב15.11 באינטל לפגישת פתיחה, גדי סקר את מטרות הפרוייקט ואת מה שנעשה בשנה האחרונה ע"י הסטודנטים שהתחילו אותו. לאחר מכן הסביר איך הוא מצפה שנמשיך את עבודתם. נתן לנו קישור לפרוייקט שלהם וקישורים נוספים לכלים שאיתם נעבוד ,במיוחד למדריך למשתמש. | |

**סטטוס הפרוייקט לחודש \_\_נובמבר\_\_\_\_\_\_**

|  |  |
| --- | --- |
| תאריך | 05/11/12 |
| סטאטוס )מה עשיתי החודש...( | קבענו פגישה עם המנחה, מחכים לאישור ממנו שיאשר ויסכים שנעשה איתו את הפרויקט |
| בעיות מיוחדות | אין |
| שינויי לו"ז | אין |
| תאריך מפגש עתידי עם המנחה | כרגע הוא בbusiness trip, מחכים לתשובה |
| תוכנית עבודה לחודש הקרוב | מחכים למנחה |
| הערות | אין |
| **סיכום פגישות עם המנחה שנעשו במהלך החודש:**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | |
|  | |

## Notes & tasks

### High-level plan

1. Ramp-up and research on Intel compiler and LibTune
2. Abstract APIs (common interface for Windows & Linux)
3. Correctness validation
4. Code cleanup – comments, include of system libraries before private, no redundant includes,

add full documentation regarding code that isn’t in use any longer

1. Run tests
2. Makefile
3. Documentation
   1. Knobs meaningful help
   2. Functions documentation
   3. User guides, Developer guide, Presentation
4. Packaging
5. Web site

### Low-level plan

|  |  |  |
| --- | --- | --- |
| # | **Task** | **Status** |
|  | **Ramp-up and research on Intel Compiler and LibTune** | ✓ |
|  | Download Intel Composer |  |
|  | Build libTune, try to run various knobs & see results | ✓ |
|  | Build VS project for LibTune | ✓ |
|  | Read “Intel Composer” API’s & find the following: | ✓ |
|  | **sin** & **cos** – math libraries, libimf | ✓ |
|  | memset & memcpy – memory libraries, libintelc | ✓ |
|  | vectorized versions, parallel versions | ✓ |
|  | Find all the corresponding functions from: libintelC, libimf | ✓ |
|  | Find Intel Windows DLL’s | ✓ |
|  | Then find the MSVC corresponding functions (msvc80, msvc90, etc) | ✓ |
|  | Gather knobs to groups, such as:  “-all” – bruteforce all knobs,  “-math”, “-mem”, “-string”, “-vec –math”, “-par –mem”, “-mem\_vec”, “-mem\_par”, “-memcpy”, “-memcmp”, “-strcpy”  **we added knobs:**   * **all** – applies all instrumentations   **we modified:**   * **math** – applies all math instrumentations | ✓ |
|  | Add a **libs** directory and put all intel libraries inside | ✓ |
|  | Fix all compilation errors of LibTune | ✓ |
|  | Comment out the knob that defines specific libraries,  e.g. intel\_mem\_funcs, and alike | ✓ |
|  | Change knobs "instrument\_\*" to \*  e.g. "-instrument\_math" to "-math" | ✓ |
|  | Change “-instrument\_math ICC” 🡪 "-instrument\_math 0/1"  And use ICC (libimf) | ✓ |
|  | ICC\_functions.cpp  Function **instrumentMathFast** , should we limit to MSVC image names or any other compiler?  **Yes, for now** Microsoft dll’s only! | ✓ |
|  | Add knob for extra user-defined images  (which images to look for functions for replacement) | ✓ |
|  | Add knob “-mkl” 0/1 | ✓ |
|  | If both instrument\_math and instrument\_mkl are ON,  Define an algorithm that checks which is better: ICC or MKL  Should we leave it for the other team ( zalik)  **Lower priority – need to check input data (how??)** | ✓ |
|  | Add knob for static/dynamic linking of the application  **Solution: look for functions in predefined DLL’s & main image(PIN’s API isMainExecutable)**  **Lower priority** | ✓ |
|  | memchr.cpp,  function MemchrWrapper,  what does it check: size/2 %4 != 0 ?  **Low priority** | ✓ |
|  | Ststr.cpp  We replaced **ffs** function by \_bit\_scan\_forward Intel intrinsics equivalent.  We need to:   1. Verify that they are equivalent 2. Look for the Microsoft equivalent and add #ifdef   To allow compile with ICC & MSVC  2)the Microsoft equivalent is \_BitScanForward or \_BitScanForward64 in the header <intrin.h>  And its works like this:  unsigned char isNoZero;  unsigned long index;  isNoZero = \_BitScanForward(&index ,0);  if(isNoZero)  cout<< index <<endl;  else  cout<< "is zero" <<endl;  <http://msdn.microsoft.com/en-us/library/wfd9z0bb(v=vs.80).aspx>  its works much like ffs but ffs returns int – the index of the bit 1-32 , if the number is 0 then ffs returns 0 and if the number is 1 ffs returns 1.(I ran it on ubuntu)  in the MS version if the number is 0 the index=0 but if the number is 1 also index=0 ;  we need to check the return value to know which is it.  We used bit\_scan\_forward OF intel compiler : basicly works like ffs(x)-1; | ✓ |
|  | Read\_ahead.cpp  // #include <unistd.h>  #define ssize\_t int  We couldn’t find the ssize\_t equivalent in windows, is that ok?  Found at : <http://msdn.microsoft.com/en-us/library/windows/desktop/aa383751(v=vs.85).aspx>   BaseTsd.h  typedef LONG\_PTR SSIZE\_T;  and for LONG\_PTR:  #if defined(\_WIN64)  typedef \_\_int64 LONG\_PTR;  #else  typedef long LONG\_PTR;  #endif  I ran sizeof function on ssize\_t in linux and SSIZE\_T in windows ,the two of them return 4  (as int and long)  So I assume ssize\_t and SSIZE\_T are the same.  Note:  We also got "size\_t" in line 30 , but it compiles because it belongs to namespace::std,  I think it comes from string.h and it`s for strings and not exactly what we need so maybe we should replace it with SIZE\_T just to be sure. | ✓ |
|  | Generalnstrumentation.cpp  Function **openLibintlc()**  We don’t load lazily, is it ok?  **RTLD\_LAZY**  Perform lazy binding. Only resolve symbols as the code that references them is executed. If the symbol is never referenced, then it is never resolved. (Lazy binding is only performed for function references; references to variables are always immediately bound when the library is loaded.)  From : <http://linux.die.net/man/3/dlopen>  Didn’t find yet a windows version of lazy library loading.  We need to add error message on failure  Verify that the replacement of type libintlcHander are ok, we changed the type from **VOID \*** to **HMODULE**  in: <http://msdn.microsoft.com/en-us/library/windows/desktop/aa383751(v=vs.85).aspx>  HMODULE trace back to void\* | ✓ |
|  | Locks.cpp  Verify that typedef long pthread\_spinlock\_t;  Is valid!  Sleep in linux is in microseconds and in windows in miliseconds , what should we do? Divide by 1000? Leave it be?  NO intel/Microsoft official pthread header!!!!  For now : disabling locks`s knob for windows.  **Nafta:** disable knob for Windows and update help and logic accordingly | ✓ |
|  | Go over the test files of Noam, and adjust them to Microsoft | ✓ |
|  | Add makefile for Noam tests | ✓ |
|  | Need to change the help menu knob | ✓ |
|  | Disclaimers (for Windows only):   1. The following list of functions will not be covered with the "-math 1" knob:   "cbrt","cbrtf", "copysignf", "copysignl", "erfcf","erff","exp2l","fdim", "log1p","log1pf","log1pl","lroundl", "nearbyintl","nextafterl","nexttoward","nexttowardf","nexttowardl","remainderl","remquol","scalblnl","scalbnl", "tgamma","tgammaf","trunc","truncl” | ✓ |
|  | **Abstract APIs** | ✓ |
|  | **Correctness** | ✓ |
|  | Correctness -math,  **Dror** – generate a program with a call to each of the functions in icc\_func\_dict, then discover all DLLs  Used by dependency walker and add them to initImageNames()  **Vladimir –**   1. Math images, **RTN\_Valid(cos)** is false for **msvcr100d.dll,** why? 2. Cos is found in main image (executable), how comes?   Dependency walker shows that it’s in **msvcr100d.dll**   1. Does PIN replace import address table entries too? 2. Why 62 functions replaced in 32-bit and only 29 in 64-bit mode?   (Files: C:\Users\ocohen11\Dropbox\Project\Resources\LibTune\64-bit functions replaced math knob.txt  And mathX86ReplacedFunc.log) | ✓ |
|  | Correctness -mkl,   1. **Nafta/Gadi** – works very slowly!   It replaces the call to **atan** with **vdAtan** which simulates **atan** with an array of 1 element  Do we always replace like that?  **Solution: Ignore –mkl for now & tests** | ✓ |
|  | Correctness , -read\_ahead,   1. **Gadi** – the instrumentRead function in **read\_ahead.cpp** should be limited to the main application image only?   **Do not limit, or search for the image name that has the read function**!   1. Change the **prototype** of **read** and **write** to be ret value **ssize\_t** (8 bytes) in unix and **int** (4 bytes) in windows   **Status**: a bit problematic, we don’t know what is the actual underlying API (\_open, open or open\_s?) in Windows (as opposed to Unix??)  **Gadi/Vladimir**:   1. Are all UNIX **read** and **close** functions go through the **read** and **close** system calls? 2. What is the underlying API call for **read** and **close** in windows?   **Solution**: Not relevant for Windows  need to do a thorough research in order to determine which read functions are worthwhile for replacement | ✓ |
|  | **Gadi**, **–mem**, is it related to the parallel versions, or just non-parallel optimized **mem** funcs?  **Answer**: it replaces the functions with Intel’s version, need to verify that it works,  also need to find out which library actually provides: **ICC/TBB/IPP**  Chosen library: ipp  See table below | ✓ |
|  | Correctness, **-parallel\_\*** | ✓ |
|  | Correctness, **-vectorized\_strchr, -vectorized\_strstr**  **Gadi**, the Intel function returns a **char const\*** whereas the MSVC/linux returns **char \***,  is it ok or do we need to duplicate the memory region?  **Solution**: fix the signatures | ✓ |
|  | Correctness, **-vectorized\_strcmp**,  **Gadi,** no implementation, Noam commented out the code.  What do you want to do? This knob is disabled for now.  **Solution**: keep this knob disabled | ✓ |
|  | **Gadi**, notice him that there are missing **memmem, memchr, memcmp, bzero**, due to not appearing in either MSVC or IPP implementation. | ✓ |
|  | Instrument functions – check for both **IsMainExecutable** and **is\_blabla function** | ✓ |
|  | Send package & disclaimers (README) for Win32, Win64 of LibTune | ✓ |
|  | **Gal Tzalik**, what is the optimal number of threads to be used by default in **–parallel**? |  |
|  | Code cleanup - comments, include of system libraries before private, no redundant includes,  add full documentation regarding code that isn’t in use any longer, remove all TODO’s | ✓ |
|  | Documentation | ✓ |
|  | Tests | ✓ |
|  | Limit the **–spinlock** knob to linux only | ✓ |
|  | **Backup** |  |
|  | Dump neat report of all functions replaced | ✓ |
|  | Refactoring | ✓ |
|  | Optimize further by passing references/pointers to utilities functions, instead of passing instances | ✓ |
|  | Make the following knobs mutually exclusive: **KnobMath, KnobMathMKL** | ✓ |

## Software Requirements Specification

### Introduction

#### Purpose

In this document we will explain how to design a Pin Tool which enable optimizations on target application and produce speedup performance of this application. We will elaborate about it`s features, interfaces ,environment and its limitations. This document intended for the client and for developers as well.

#### Scope

The product - LibTune , will produce optimizations on target applications using Intel Pin.

Libtune provides Intel optimized implementations and custom implementations for library functions. The goal is to show speedup in executing application with Intel optimized function from Intel libraries such as IPP, MKL ,TBB on Windows platforms.

Some optimizations will not be consistent across platforms (Linux, Windows 32-bit/64-bit)

#### Definitions, Acronyms, and Abbreviations

**Pin** - Pin is a dynamic binary instrumentation framework for the IA-32 and x86-64 instruction-set architectures that enables the creation of dynamic program analysis tools.

**Pin Tool** - The tools created using Pin, called Pin tools, can be used to perform program analysis on user applications in Linux and Windows.

**Probed Mode** – *Pin`s mode -* Probe mode is a method of using Pin to insert probes at the start of specified routines. A probe is a jump instruction that is placed at the start of the specified routine. The probe redirects the flow of control to the replacement function. Before the probe is inserted, the first few instructions of the specified routine are relocated

***Jit* Mode** - *just-in-time compiler to insert instrumentation into running applications, the JIT compiler recompiles and instruments chunks of binary code prior to executing them and saves the recompiled code in a code cache (to allow Pin to reuse compiled code).*

**LibTune** - LibTune is a Pin Tool , LibTune is a Dynamic Binary Instrumentation tool that allows the user to optimize an already compiled application, without changing the source code. That is, the optimizations offered by this tool are related to load-time and link-time optimizations.

***ICC*** *-* Intel C++ Complier. Contain built-in libraries (such as a math library).

***MSVC*** *-* MicroSoft Visual C++ (MS) , refers to Microsoft Compiler.

***MKL*** *-* Intel Math Kernel Library.

***IPP*** *-* Intel - Integrated Performance Primitives library

#### References

* Pin – a dynamic binary instrumentation tool: <http://www.pintool.org>
* Windows API documentation:

<http://msdn.microsoft.com/en-us/library/windows/desktop/>

* Intel Compiler Suite:

<http://software.intel.com/en-us/intel-compilers>

* Intel Integrated Performance Primitives (IPP):

<http://software.intel.com/en-us/articles/intel-integrated-performance-primitives-documentation/>

* Intel® Math Kernel Library (MKL):

<http://software.intel.com/en-us/articles/intel-mkl-kb-home/>

* Dependency Walker:

<http://www.dependencywalker.com/>

#### Overview

This document contains two more sections.

In the following section we will give a general description of the product , Product Perspective,Product Functions and General Constraints.

In the last section we will demonstrate the requirements that are used to guide the project’s

software design, implementation, and testing.

### General Description

#### Product Perspective

The LibTune project was built and designed for Linux only. It was never built or tested on Windows platform. Our first objective was to port the implementation of LibTune to Windows and validating the correctness of the tool & comparing the behavior of the tool on Windows vs. Linux. The project was written in C++.

#### Product Functions

LibTune preforms the following optimizations:

**math optimizations** – replacing original functions ( most of cmath functions) with ICC and MKL functions with the same name and signature .

**memory optimizations** - replacing original functions ( memset, memcopy, malloc, free) with IPP optimized versions .

**parallel memory optimizations** - replacing original functions ( memset, memcopy) with parallel custom versions .

**read ahead optimization** - read() and close() low-level I/O functions will be replaced by custom implementation.

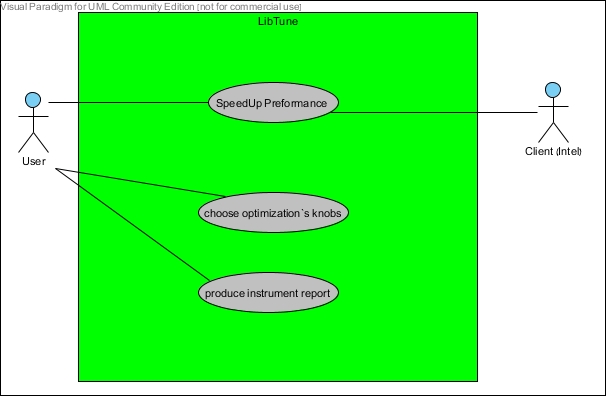
**Vectorized string optimizations** – replacing original string functions (strchr(),strstr() ,strlen() ) will be replaced with the custom vectorized version.

#### General Constraints

For reviewing constraints and limitations please refer to "Technical difficulties" section in the main document.

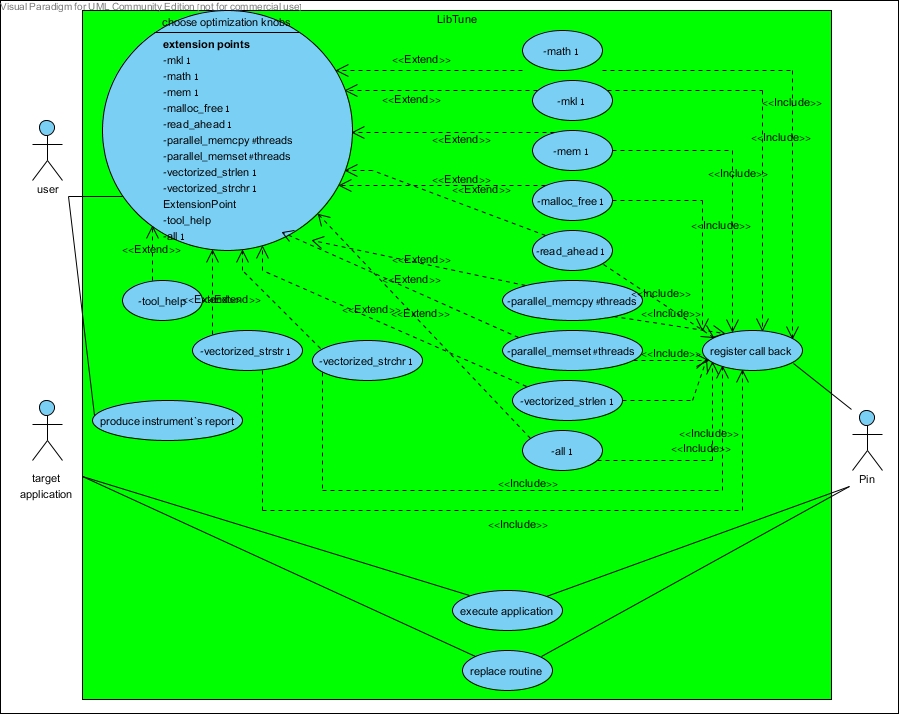
### Specific Requirements

#### Business Level Use Case Diagram



|  |  |  |  |
| --- | --- | --- | --- |
| BucId | Use case name | Actor | Description |
| BUC-1 | SpeedUp performance | user,client | the interest of the user is to achieve better performance for his application and the client`s interest is to demonstrate how the application will run better with his compiler`s libraries and CPU. |
| BUC-2 | choose optimization`s knobs | user | the user will choose which optimizations he wants to run on his application, he can choose one knobs, all knobs or any combination of the knobs |
| BUC-3 | produce instrument report | user | after the application completed its run, the user will receive a log file . in this file he can see all the function that was replaced and in which dll they were found. |

#### System level use case diagram



|  |  |  |  |
| --- | --- | --- | --- |
| SucId | Use case name | Actor | Description |
| SUC-1 | choose optimization`s knobs | user | the user will choose which optimizations he wants to run on his application, he can choose one knobs, all knobs or any combination of the knobs |
| SUC-2 | produce instrument`s report | user | after the application completed its run, the user will receive a log file . in this file he can see all the function that was |
| SUC-3 | -tool\_help | user | choosing this knob will report to console about the verity of optimization`s knobs and the parameters they get. |
| SUC-4 | -math 1 | user | preforms optimizations on mathematical functions such as cos,tan… using ICC library functions |
| SUC-5 | -mkl 1 | user | preforms optimizations on mathematical functions such as cos,tan… using MKL library functions |
| SUC-6 | -mem 1 | user | Choosing this knob will apply memory function optimizations . Replacing original functions with IPP library functions. |
| SUC-7 | -malloc\_free 1 | user | Replacing original malloc and free functions with IPP ippsMalloc ,ippsFree functions. |
| SUC-8 | -read\_ahead 1 | user | Replacing original read() ,close() functions with custom functions. |
| SUC-9 | -parallel\_memcpy #threads | user | by inserting number of threads . this knob invokes memcpy function divided into #threads jobs, working prallely. |
| SUC-10 | -parallel\_memset #threads | user | by inserting number of threads . this knob invokes memset function divided into #threads jobs, working prallely. |
| SUC-11 | -vectorized\_strlen 1 | user | Replacing original strlen() functions with optimized custom functions. |
| SUC-12 | -vectorized\_strchr 1 | user | Replacing original strchr() functions with optimized custom functions. |
| SUC-13 | -vectorized\_strstr 1 | user | Replacing original strstr() functions with optimized custom functions. |
| SUC-14 | -all | user | invokes all knobs above . |
| SUC-15 | execute application | target application, Pin | Pin loads the target application |
| SUC-16 | register callback | Pin | At “image load” callback event, Pin invokes all the subscribed callbacks and they decide which functions to instrument, and do the replacement via Pin |
| SUC-17 | replace routine | target application, Pin | At “image load” callback event, Pin invokes all the subscribed callbacks and they decide which functions to instrument, and do the replacement via Pin. |

#### Textual Use cases

|  |  |
| --- | --- |
| Criterion | ID: SUC-1 Name: choose optimization`s knobs |
| Pre-conditions | PIN has started in probe mode and an application has supplied |
| Post-conditions | libtune registered the function`s call back with Pin |
| Triggers | on libtune load |
| branching off | if, knob is unknown output to console - tool\_help |

|  |  |
| --- | --- |
| Criterion | ID: SUC-2 Name: produce instrument`s report |
| Pre-conditions | optimization knobs was provided and Pin found function`s call to replace |
| Post-conditions | after the application completed its run, the user will receive a log file . in this file he can see all the function that was |
| Triggers | application completed execution |
| branching off | if no replacement has happened , log file is empty |

|  |  |
| --- | --- |
| Criterion | ID: SUC-3 Name: -tool\_help |
| Pre-conditions | command line string given, Pin is running |
| Post-conditions | report to console about the verity of optimization`s knobs and the parameters they get. |
| Triggers | knob is chosen |

|  |  |
| --- | --- |
| Criterion | ID: SUC-4 Name: -math 1 |
| Pre-conditions | command line string given, Pin is running, ICC dll is provided |
| Post-conditions | Mathematical functions replaced with ICC library functions |
| Triggers | knob is chosen |
| branching off |  |
| extended Uc | SUC-1 |
| included Uc | SUC-16 |

|  |  |
| --- | --- |
| Criterion | ID: SUC-5 Name: -mkl 1 |
| Pre-conditions | command line string given, Pin is running, mkl dll is provided |
| Post-conditions | Mathematical functions replaced with MKL library functions |
| Triggers | knob is chosen |
| extended Uc | SUC-1 |
| included Uc | SUC-16 |

|  |  |
| --- | --- |
| Criterion | ID: SUC-6 Name: -mem 1 |
| Pre-conditions | command line string given, Pin is running IPP dll is provided |
| Post-conditions | memset, memcpy functions replaced with IPP library functions |
| Triggers | knob is chosen |
| extended Uc | SUC-1 |
| included Uc | SUC-16 |

|  |  |
| --- | --- |
| Criterion | ID: SUC-7 Name: -malloc\_free 1 |
| Pre-conditions | command line string given, Pin is running IPP dll is provided |
| Post-conditions | malloc and free functions replaced with IPP ippsMalloc ,ippsFree functions. |
| Triggers | knob is chosen |
| extended Uc | SUC-1 |
| included Uc | SUC-16 |

|  |  |
| --- | --- |
| Criterion | ID: SUC-8 Name: -read\_ahead 1 |
| Pre-conditions | command line string given, Pin is running |
| Post-conditions | read() ,close() functions replaced with custom functions. |
| Triggers | knob is chosen |
| extended Uc | SUC-1 |
| included Uc | SUC-16 |

|  |  |
| --- | --- |
| Criterion | ID: SUC-9 Name: -parallel\_memcpy |
| Pre-conditions | command line string given, Pin is running |
| Post-conditions | memcpy function divided into #threads jobs, working prallely. |
| Triggers | knob is chosen |
| branching off | If parallelize is not worthy (amount of bytes to copy <300,000) do not invoke parallel implementation. Original memcpy will run or IPP version (if –mem were also chosen)  If set threads are busy , Original memcpy will run or IPP version (if –mem 1 were also chosen) |
| extended Uc | SUC-1 |
| included Uc | SUC-16 |

|  |  |
| --- | --- |
| Criterion | ID: SUC-10 Name: -parallel\_memset |
| Pre-conditions | command line string given, Pin is running |
| Post-conditions | memset function divided into #threads jobs, working prallely. |
| Triggers | knob is chosen |
| branching off | If parallelize is not worthy (amount of bytes to set <300,000) do not invoke parallel implementation. Original memset will run or IPP version (if –mem were also chosen)  If set threads are busy , Original memset will run or IPP version (if –mem 1 were also chosen) |
| extended Uc | SUC-1 |
| included Uc | SUC-16 |

|  |  |
| --- | --- |
| Criterion | ID: SUC-11 Name: -vectorized\_strlen 1 |
| Pre-conditions | command line string given, Pin is running |
| Post-conditions | strlen function replaced with custom and optimized strlen function |
| Triggers | knob is chosen |
| extended Uc | SUC-1 |
| included Uc | SUC-16 |

|  |  |
| --- | --- |
| Criterion | ID: SUC-12 Name: -vectorized\_strchr 1 |
| Pre-conditions | command line string given, Pin is running |
| Post-conditions | strchr function replaced with custom and optimized strchr function |
| Triggers | knob is chosen |
| extended Uc | SUC-1 |
| included Uc | SUC-16 |

|  |  |
| --- | --- |
| Criterion | ID: SUC-13 Name: -vectorized\_strstr 1 |
| Pre-conditions | command line string given, Pin is running |
| Post-conditions | strstr function replaced with custom and optimized strstr function |
| Triggers | knob is chosen |
| extended Uc | SUC-1 |
| included Uc | SUC-16 |

|  |  |
| --- | --- |
| Criterion | ID: SUC-14 Name: -all |
| Pre-conditions | command line string given, Pin is running |
| Post-conditions | all optimization`s knobs were invoked and all supported function has been replaced |
| Triggers | knob is chosen |
| extended Uc | SUC-1 |
| included Uc | SUC-16 |

|  |  |
| --- | --- |
| Criterion | ID: SUC-15 Name: execute application |
| Pre-conditions | command line string given, Pin is running , application had provided |
| Post-conditions | application is running on Pin VM . |
| Triggers | command line string given. |

|  |  |
| --- | --- |
| Criterion | ID: SUC-16 Name: register callback |
| Pre-conditions | knobs were chosen Pin is running , application had provided |
| Post-conditions | all function`s call back which were chosen is registered with Pin . |
| Triggers | image load |

|  |  |
| --- | --- |
| Criterion | ID: SUC-17 Name: replace routine |
| Pre-conditions | knobs were chosen Pin is running , application had provided |
| Post-conditions | Pin invokes all the subscribed callbacks and they decide which functions to instrument, and do the replacement via Pin. |
| Triggers | image load |

### Non-Functional Requirements

#### Performance

##### Math optimizations

we have noticed significant speedup on some applications using trigonometric functions. Therefore, these optimizations are recommended to be used by the tool, although we cannot guarantee full coverage of math libraries (due to the problems we experienced on Windows).

##### Memory optimizations

For malloc and free functions we saw improvement, 1.12x speedup. This happens when certain conditions are met, such as when the application is memory bounded and malloc is used extensively.

For memset we saw ~ 1.2x (up to) speedup, and for memcpy we saw ~ 1.4x (up to) speedup.

For parallel memory operations (using the -parallel knobs) we saw significant slowdown, and we do not recommend using them, or at least do more research before using these knobs (perhaps they are good for some applications).

##### Read ahead optimization

Read ahead optimization is not recommended to be used, because of the limitation (see limitations sections for more information), therefore we do not recommend to use it.

##### Vectorized string optimizations

On processors with Intel® SSE technology, the vectorized string operations can benefit a lot due to a good exploitation of the vector registers in the architecture, and we definitely recommend using them.

#### Reliability

Libtune should ensure that no harm is done to the instrumented application.

The original files stays intact and are not modified .

each replacement function is expected to behave exactly like the original one in terms of effect on memory, variables ,output and program state .

#### Maintainability

We provide extensive developer`s manual for the product`s developer successor.

Note : Sequence Diagrams and Data Flow Diagrams (DFD) are provided in the main document