|  |
| --- |
| École de technologie Supérieure  Département de génie logiciel et des TI |
| Yarn  A Speculative Multithreading System |
| Vision |
|  |
| **Rémi Attab** |
| **6/22/2011** |

|  |
| --- |
|  |

Table of Contents

[List of tables 3](#_Toc296537623)

[List of figures 3](#_Toc296537624)

[1. Introduction 4](#_Toc296537625)

[1.1 Objectives 4](#_Toc296537626)

[1.2 Scope 4](#_Toc296537627)

[2. Product positioning 4](#_Toc296537628)

[2.1 Problem statement 4](#_Toc296537629)

[2.2 Product positioning 4](#_Toc296537630)

[3. Description of the users 5](#_Toc296537631)

[3.1 Target market 5](#_Toc296537632)

[3.2 User description 5](#_Toc296537633)

[3.3 Main user needs 5](#_Toc296537634)

[B1. Parallelizes sequential code 5](#_Toc296537635)

[B2. Easy to use 5](#_Toc296537636)

[B3. Computation must be coherent 5](#_Toc296537637)

[B4. Is fast 5](#_Toc296537638)

[B5. Is portable 6](#_Toc296537639)

[3.4 Alternatives 6](#_Toc296537640)

[4. Global product overview 6](#_Toc296537641)

[4.1 Product perspective 6](#_Toc296537642)

[4.2 Main advantages 7](#_Toc296537643)

[4.3 Licensing 8](#_Toc296537644)

[5. Product characteristics 9](#_Toc296537645)

[C1.1 Loop-based speculation 9](#_Toc296537646)

[C1.2 Function-based speculation and return value prediction 9](#_Toc296537647)

[C2.1 Automatic instrumentation of speculative loops 9](#_Toc296537648)

[C2.2 Automatic instrumentation of speculative function calls 9](#_Toc296537649)

[C2.3 Automatic instrumentation of function calls 9](#_Toc296537650)

[C2.4 Automatic instrumentation of dependencies 9](#_Toc296537651)

[C3.1 Validate accesses to epoch-shared dependencies 9](#_Toc296537652)

[C3.2 Validate accesses to aligned words dependencies 9](#_Toc296537653)

[C3.3 Validate accesses to un-aligned addresses dependencies 10](#_Toc296537654)

[C3.4 Validate accesses to an address range dependencies 10](#_Toc296537655)

[C3.5 Validate accesses to IO dependencies 10](#_Toc296537656)

[C3.6 Error recovery 10](#_Toc296537657)

[C4.1 The instrumented code should execute faster than the sequential code 10](#_Toc296537658)

[C4.2 Optimization of the instrumentation output 10](#_Toc296537659)

[C5.1 Adapt to the number of available processors 10](#_Toc296537660)

[C5.2 Support for 64 bit processors 10](#_Toc296537661)

[C5.3 Support for 32 bit processors 11](#_Toc296537662)

[C5.4 Support for various operating systems 11](#_Toc296537663)

[6. Constraints 11](#_Toc296537664)

[7. Attributes of the characteristics 12](#_Toc296537665)

# List of tables

[Table 1 - Benefits matrix 7](#_Toc296536764)

[Table 2 - Attributes of the characteristics 12](#_Toc296536765)

# List of figures

[Figure i - System overview 6](#_Toc296536766)

# 1. Introduction

## 1.1 Objectives

This vision document describes a high level view of the yarn project and the context in which it will be introduced. It will describe its primary users and the market that we aim to introduce it to. It will also describe the primary needs of the users and the characteristics of the system that will address them.

## 1.2 Scope

This document will limit itself to the description of the two main components that make up the proposed solution. The context and environment in which the solution will be integrated will also be described. This document doesn't include a thorough description and analysis of competing tools or of existing techniques. This document will briefly describe the behaviour of the system but will not describe detailed requirements.

# 2. Product positioning

The objectives for the proposed solution are:

* To fully utilize modern multi-core processors by automatically parallelizing sequential programs.
* To quickly and easily parallelize a program with little or no user intervention.
* To ensure the correctness of the parallelisation process.
* To allow the user to guide and optimize the results of the parallelization process.
* To have a portable parallelization scheme.

## 2.1 Problem statement

**The problem**: Manually turning a sequential program into a parallel program is a difficult, time-consuming and error-prone task.

**The impact:** Manually parallelized programs can be unstable and sometimes slower than their sequential counter-part.

**Best-case solution:** A tool that can quickly and automatically generate fast parallel code with no user intervention. The tool should also warn the user in the case where no speed-up is possible for a given program.

## 2.2 Product positioning

**For who:** Any software developer working on a sequential program.

**The system is:** A tool that can be used to transform a sequential program into a parallel program. The tool will also use a runtime component to ensure the correctness of the transformation.

**The system is not:** A tool to optimize or manipulate a program that is already parallel.

**Our product:** Should integrate itself easily into existing development environments.

# 3. Description of the users

## 3.1 Target market

The proposed solution will mainly be targeted at programs that executes on a client's machine as opposed to server side programs. This focus is because server side programs are, in most cases, already parallel programs. The solution can't improve and already parallel program.

Special attention should also be accorded to legacy programs. These programs are often very large sequential programs that are very difficult to parallelize manually due to their accumulated size and complexity. The proposed solution is especially well suited to tackle these types of applications and could provide very good value to their maintainers.

## 3.2 User description

The target users for the solution are software developers working on the programs described in the 3.1 section. These developers will only be required to know very basic knowledge about software parallelization in order to use the solution.

## 3.3 Main user needs

The section describes the primary user needs that our project aims to fill.

### B1. Parallelizes sequential code

**Priority:** Critical

**Preoccupation:** To fully utilize modern parallel hardware to speed up a sequential program.

**Current solution:** Manually parallelize the code using one of the available tools.

**Proposed Solution:** Introduce a new tool that parallelizes code using a speculative multithreading scheme.

### B2. Easy to use

**Priority:** Important

**Preoccupation:** The parallelization process must simple.

**Current solution:** Manual parallelization is complicated and error prone process which can be somewhat simplified by using available tools.

**Proposed Solution:** A speculative multithreading can function with little or no user intervention.

### B3. Computation must be coherent

**Priority:** Critical

**Preoccupation:** The result of the parallel computation must be the same as the sequential computation.

**Current solution:** Manual analysis of the code to introduce synchronization primitives.

**Proposed Solution:** The proposed solution introduces automatic dependency checking at runtime that ensures that the computation remains coherent.

### B4. Is fast

**Priority:** Important

**Preoccupation:** The parallel computation must execute faster than its sequential version.

**Current solution:** Manually parallelize the code and execute performance tests followed by profiling and optimization.

**Proposed Solution:** The parallelization of the program should ensure that in most cases the program performs faster. Tools will be introduced to diagnose and correct performance problems.

### B5. Is portable

**Priority:** Low

**Preoccupation:** The parallelization should work across various processor architecture and operating system.

**Current solution:** Use cross-platform tools.

**Proposed Solution:** The implementation of the speculative multithreading system will be cross-platform.

## 3.4 Alternatives

There is currently only one known alternative implementation of a speculative multithreading system that has been released for general use. The *libspmt*[[1]](#footnote-1) project was created Christopher J.F. Picket and Clark Verbugge at the University of McGill as part of a thesis project. It targets the Java platform and implements a function based speculation scheme. More research is required to determine how viable an alternative it is to our project.

# 4. Global product overview

## 4.1 Product perspective



Figure - System overview

The yarn project is composed of two components: a runtime component (libyarn) and a compiler component (yarnc). The yarnc will be in charge of analyzing a given program for potential parallelization opportunities. It will then generate instrumented code with calls to the runtime component. The runtime component will be part of the final executable and will be in charge of running the code in parallel and of keeping the computation coherent.

## 4.2 Main advantages

This section will map the main characteristics of the project described in section 5 to the needs that they fulfill which were described in section 3.3.

|  |  |
| --- | --- |
| User benefits | Characteristics |
| B1. Parallelizes sequential code | C1.1 Loop-based speculation |
| C1.2 Function-based speculation and return value prediction |
| B2. Easy to use | C2.1 Automatic instrumentation of speculative loops |
| C2.2 Automatic instrumentation of speculative function calls |
| Automatic instrumentation of function calls |
| C2.4 Automatic instrumentation of dependencies |
| B3. Computation must be coherent | C3.1 Validate accesses to epoch-shared dependencies |
| C3.2 Validate accesses to aligned words dependencies |
| C3.3 Validate accesses to un-aligned addresses dependencies |
| C3.4 Validate accesses to an address range dependencies |
| C3.5 Validate accesses to IO dependencies |
| C3.6 Error recovery |
| B4. Is fast | C4.1 The instrumented code should execute faster than the sequential code |
| C4.2 Optimization of the instrumentation output |
| B5. Is portable | C5.1 Adapt to the number of available processors |
| C5.2 Support for 64 bit processors |
| C5.3 Support for 32 bit processors |
| C5.4 Support for various operating system |

Table - Benefits matrix

## 4.3 Licensing

Every documents and the entire source code for this project will be open sourced using the FreeBSD license. The license is reproduced here:

Copyright 2011 Rémi Attab. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY Rémi Attab ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL Rémi Attab OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The views and conclusions contained in the software and documentation are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of Rémi Attab.

# 5. Product characteristics

## C1.1 Loop-based speculation

Loop-based speculation takes executes each of its iterations in parallel. The characteristics C3.1, C3.2, C3.3, C3.4 and C3.5 describe how the computation is kept coherent.

## C1.2 Function-based speculation and return value prediction

The general concept is to delegate function calls to a new thread that is executed in parallel while the main thread continues executing after predicting the return value of the function. The characteristics C3.1, C3.2, C3.3, C3.4 and C3.5 describe how the computation is kept coherent.

## C2.1 Automatic instrumentation of speculative loops

This characteristic analyzes a program for potentially parallelizable loops and adds the required calls to the runtime library to make it run in parallel.

This characteristic is only applicable to loop-based speculation described in C1.1.

## C2.2 Automatic instrumentation of speculative function calls

This characteristic analyzes a program for function calls that could be executed while the current function is executed speculatively. To do that, the system will need to predict the return value of the called function. The user can add pragma directives into the code to guide the analysis of the code.

This characteristic is only applicable to function-based speculation described in C1.2.

## C2.3 Automatic instrumentation of function calls

This characteristic analyzes a program for function calls during that are to be executed as part of the current epoch. This differs from C2.2 because no new epochs are generated for the function call. The function being called will be instrumented so that any modification made to shared dependencies will be tracked. Alternatively, the user can add pragma directives in the code to indicate which dependencies are modified by the function.

This characteristic is applicable to both loop-based speculation described in C1.1 and function-based speculation described in C1.2

## C2.4 Automatic instrumentation of dependencies

This characteristic analyzes a program for dependencies shared between the epoch-shared dependencies. These dependencies will be instrumented so that every reads and writes are tracked and checked for coherence.

This characteristic is applicable to both loop-based speculation described in C1.1 and function-based speculation described in C1.2

## C3.1 Validate accesses to epoch-shared dependencies

Every writes to an epoch-shared dependency should be tracked and buffered. Every reads to an epoch-shared dependency should be tracked and return the value of an earlier buffered write or the current value in memory if no previous write had occurred. After each writes, every reads that may have occurred in a later epoch will trigger a dependency violation.

## C3.2 Validate accesses to aligned words dependencies

Modern compilers will naturally align every word so that they can be easily accessed by the processor. These are the simplest kinds of dependency types that don't require any special support.

## C3.3 Validate accesses to un-aligned addresses dependencies

Un-aligned addresses occur when using variable that are smaller or bigger than the natural word size of a processor. For example, a string can usually fit multiple characters within a single word or a floating point value can require multiple words to store. The aim is to provide support for these data types. Note that this affects the portability that is discussed in B5.

## C3.4 Validate accesses to an address range dependencies

Many common operations on arrays will apply a simple transformation on a large range of addresses. For example, after allocating an array, it is customary to initialize every element with the value 0. This can take the form of a call to the standard c function memset or a simple for-loop. The aim is to use more efficient data structures (a radix tree) to quickly handle these cases.

## C3.5 Validate accesses to IO dependencies

IO can take many shapes and can be quite difficult to support. The aim is to provide support for common types of IO like files and network sockets. This can be implemented by doing only a single physical read on the IO device. Every subsequent reads will be directed to a buffer instead.

## C3.6 Error recovery

If an error occurs during the parallel execution, the runtime component should recover by executing the original sequential code after the point of the last commit. For example, if a for-loop is to be executed on the variable i going from 0 to 100 and that an error occurs after the value 50 was committed on i, the sequential for-loop should resume with i equals to 51.

## C4.1 The instrumented code should execute faster than the sequential code

The runtime component will keep track of the speculative execution time and stop the speculative execution if it is taking too long. The runtime component will then resume the execution of the sequential code. This can be helped by the profiling information detailed in C4.2.

## C4.2 Optimization of the instrumentation output

The system will provide tools to help guide and profile the code instrumentation process to ensure that every instrumented sections of code don't hurt performances. This will take the form of a special profiling mode which will aim to gather lots of runtime information. This information can then be used by the users to guide the instrumentation process using pragma directives. It can also be fed back into the code instrumentation process to improve the code analysis.

## C5.1 Adapt to the number of available processors

The runtime component should automatically make full use of all available physical processors. Note that hyper-threaded processors can hurt performances since they don't run in parallel. The runtime component should ignore these processors.

## C5.2 Support for 64 bit processors

The native implementation of the system should not require any special consideration because the development of the system will take place on 64 bit systems.

## C5.3 Support for 32 bit processors

Special care should be used on 32 bit processors when dealing with floating point variables that uses the IEEE754 standard. These floating point values will not fit within a single word and its support will rely on characteristic C3.3.

## C5.4 Support for various operating systems

We currently aim to support the most popular operating systems which currently includes Linux based distributions and all versions of Windows starting from Windows XP.

# 6. Constraints

This section describes the constraints that are imposed for the development of this project.

* **CON01** - The compiler component must be developed using the LLVM compiler.
* **CON02 -** The compiler component must be developed using the C++ language.
* **CON03 -** The runtime component must be developed using the C language.
* **CON04 -** The runtime component must only used commonly available library.

# 7. Attributes of the characteristics

The following table qualifies each characteristic with attributes to help in the pacification of their implementation.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Characteristics | State | Benefits | Effort | Risks | Priority |
| C1.1 Loop-based speculation | Now | High | Medium | High | High |
| C1.2 Function-based speculation and return value prediction | Later | High | High | High | Low |
| C2.1 Automatic instrumentation of speculative loops | Now | High | Medium | Medium | High |
| C2.2 Automatic instrumentation of speculative function calls | Later | High | High | High | Medium |
| C2.3 Automatic instrumentation of function calls | Later | Medium | High | High | Medium |
| C2.4 Automatic instrumentation of dependencies | Now | High | Low | Low | High |
| C3.1 Validate accesses to epoch-shared dependencies | Now | High | Low | Low | High |
| C3.2 Validate accesses to aligned words dependencies | Now | High | Low | Low | High |
| C3.3 Validate accesses to un-aligned addresses dependencies | Later | Low | Medium | Medium | Low |
| C3.4 Validate accesses to an address range dependencies | Later | Medium | Medium | Medium | Low |
| C3.5 Validate accesses to IO dependencies | Later | High | High | High | Low |
| C3.6 Error recovery | Now | High | Low | Low | High |
| C4.1 The instrumented code should execute faster than the sequential code | Later | Medium | Medium | High | Medium |
| C4.2 Optimization of the instrumentation output | Later | Medium | High | Medium | Low |
| C5.1 Adapt to the number of available processors | Now | Medium | Low | Low | High |
| C5.2 Support for 64 bit processors | Now | High | Low | Low | Medium |
| C5.3 Support for 32 bit processors | Later | Medium | Medium | Medium | Low |
| C5.4 Support for various operating systems | Later | High | Medium | Medium | High |

Table - Attributes of the characteristics

1. https://svn.sable.mcgill.ca/sable/spmt/libspmt [↑](#footnote-ref-1)