Thesis Title

Author Name

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department
Knossou Av., P.O. Box 2208, Heraklion, GR-71409, Greece

Thesis Advisors: Prof. *First*, Dr. *Second*

Optionally: This work was partially supported by **fill in sponsor name**

University of Crete
Computer Science Department

Your Title

Thesis submitted by
Author Name
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: ______________________
Author Name

Committee approvals: ______________________
Name of first member
Assistant Professor, Thesis Supervisor

____
Name of second member
Associate Professor, Committee Member

____
Name of third member
Professor, Committee Member

Departmental approval: ______________________
Name of Director of Graduate Studies
Professor, Director of Graduate Studies

Heraklion, December 2012

Abstract

In this work ...

Περίληψη

Στην εργασία αυτή ...

Acknowledgements

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We present an implementation of the future programming model for distributed memory, using MPI-2's one-sided communication.

1.1 Motivation

Parallel computing has been mpla mpla mpla. The two most dominant and widely used programming models are threads and message passing. Threads are used on shared memory machines and require locking (error prone) while message passing can be used on either shared or distributed memory. Describe message passing, say its tough, talk about one sided communication, refer to ARMCI, ARMI, Charm++, Global address space languages, MPI-2 etc. Say that mpi is widely used and is implemented on most machines, thus we want to check out its one sided interface, which has not received acceptance due to (?) (check all that stuff I've read about how tought it is, mpla mpla). Also need to mention the future interface, and why we use it. Ease o programming, exposing irregular patterns(?).

Note:global arrays guys have already made arguments about one sided comm of mpi-2

1.2 Background

1.2.1 Futures

Background on futures, mention languages that implement it as well as std and boost in C++. Example code and explanation.

1.2.2 MPI one-sided communication

Maybe mention again thins from intro. Explain the interface (windows, epochs, put, get). Maybe a small example.

1.3 Related Work

RMI, RMC, HPX, STAPL's comm library

1.4 Related Work

Chapter 2

Design and Implementation

We have implemented the distributed futures using the one-sided mpi library.

2.1 Futures Interface

We replicate the futures interface from the C++ std::future library, with the only difference being that the the function being called must be a functor object. Fiture (ref) shows a recursive implementation of the fibonacci function using our future library. The user needs to create a functor, which must be serializable*(footnote here about boost::serialization), and use the macro FUTURES_ EXPORT_ FUNCTOR(async_ function<fib, int>) to expose the functor object to the serialization library. Note that the argument to the macro command is always async_ function<F, Args...>, where F is functor class and Args are the argument types, of any arbitary number, that are required by the overloaded call method of the functor F. A call to the async(F, Args...) function, where F is a functor object and Args is any number of arguments, will send the functor object to an available process or execute the functor directly, if no such process is found (see SECTION for details). The async function returnd a Future object which can be used by the process that called the async function to retrieve the value. In order to retrieve the value, the owner of the future needs to call the get() method. This method is blocking, so calling it will cause the process to block until the value of the future becomes available. Alternatively, the future owner can call the is_ ready() method, which is not blocking, to check if the value can be retrieved.

2.2 Communication

Overall description of how stuff work, logic behind async and future value retrieval, more details of each module at each section. Figure of program flow

2.3 Memory allocation

Explain the implementation Maybe figures...hmmm check Modern operating systems book

2.4 Scheduler

Explain implementation Maybe say alternatives (will seem weak if we do not support why we do what we do) Keep in mind that scheduling and Memory allocator are pretty standard stuff, proof of concept

Chapter 3

Methodology

General discussion . . .

3.1 AA

3.2 BB

3.3 CC

3.4 DD

3.5 EE

3.6 FF

Chapter 4

Evaluation

General discussion ...

4.1 AA

4.2 BB

4.3 CC

4.4 DD

4.5 EE

4.6 FF

Chapter 5

Comparison

Compare your work . . .

5.1 AA

5.2 BB

5.3 CC

5.4 DD

5.5 EE

5.6 FF

Chapter 6

Conclusions and Future Work