Ordered Read Write Locks for Multicores and Accelarators

Mariem Saied

INRIA & ICube
Strasbourg, France

mariem.saied@inria.fr

ORWL, Ordered Read-Write Locks

An inter-task synchronization model for data-oriented parallel algorithms

- A lock mechanism that can handle data-dependencies between threads
- A new API for resource centric parallel programming
- ORWL particularly targets iterative computations

ORWL, Ordered Read-Write Locks

Features

- A waiting queue with FIFO policy for each resource
- Distinction between write blocks (exclusive) and read blocks (inclusive)
- An explicit association of a task with application data
- Distinction between post and acquire
- Distinction between locks and lock handles

ORWL, Ordered Read-Write Locks

Typical Sequence

- Request : Insert a request in the waiting queue of the resource
- Acquire : When it becomes necessary, the process is blocked until the resource is acquired
- Release : The resource is released to grant access to other requests

Library specific to iterative computations

- With a release, post a new request on the handle of the resource for the next iteration

ORWL, Ordered Read-Write Locks

Properties

- deadlock-freeness
- liveness
- equity
- expressiveness

ORWL, Ordered Read-Write Locks

Data layout

- Data split into blocks
- Each block is considered as an ORWL data location
- One POSIX thread is associated to each block.

Optimized layout

- Enhanced version of block-layout with well-defined boundary relation between blocks
- A main task that performs the computation is defined
- The export of frontier data is ensured via sub-tasks.

ORWL, Initialization phase

- Requirement to define the access scheme between tasks and resources and to attribute initial priorities in the FIFO
- The description of the principle structure of the program is explicit and static
- A simplified initialization usage mode adapted to iterative computing algorithms is suggested :
    - All resources and their access scheme are specified in an initial phase of the application.
    - The access scheme implicitely synchronizes the tasks during the computation phase.
    - Subsequent computation phases are guaranteed to be deadlock-free and fair

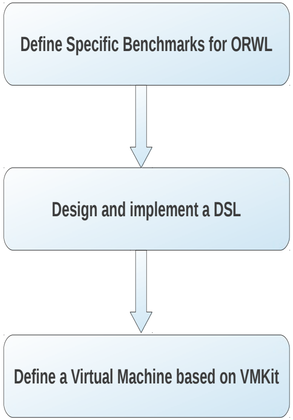
Thesis Goals

Build an abstracted execution platform for ORWL relying on VMKit

- Automate the specification of the access relation between tasks and resources and optimised attribution of initial priorities
- Step-wise transformation of ORWL to build upon specific VMKit features
- Improve ORWL in the context of multicores in particular on NUMA architectures.
- Benchmarking the implementations by means of rigurous experiments.

Followed Approach

- Define the domain
- Design the DSL respecting domain semantics
- Concepts to be included : FIFOs, ...
- Define tasks for each iteration
- Tasks priorities
- Task/Resource


Define Specific Benchmarks for ORWL
Design and implement a DSL
Define a Virtual Machine based on VMKit

ORWL Benchmark types

ORWL Benchmarks

HPC Benchmarks : Matrix Processing LINPACK (Livermore 23)

Data-flow graphs Benchmarks

MapReduce Benchmarks

State of the art {Domain Specific Languages }

Bossa : Framework enabling the implementation, deploying and management of process scheduler hierarchies.

- includes DSL for schedular implementation and verification
- provides high level abstractions :
  - process attributes, process states, process lists and events.
- Choices in the the design and implementation of the DSL :
  - Absence of pointers and impossiblity of defining infinite loops in order to provide safety guarantees
  - Choice of JIT compiler in order to guarantee flexible and efficient implementation for Bossa.

Lawall, Julia L., Gilles Muller, and Hervé Duchesne. "Invited application paper : language design for implementing process scheduling hierarchies." Proceedings of the 2004 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation. ACM, 2004.

State of the art {MapReduce, Metis }

MapReduce

- Programming model for data parallel programs hiding synchronization and parallel task management
- Intended for applications that can fit in a pair of Map and Reduce functions.
    - Three phases : Map, Reduce, Merge.

Metis

MapReduce library for multicore processors

- Compromise intermediate data structure : a hash table with a b+tree in each entry
- Appropriate for applications wich have a large number of intermediate key/value pairs and a low amount of computation

State of the art {Data Flow Graphs }

X-Stream : Edge-centric Graph Processing using Streaming Partitions

- edge-centric system
- relies on sequential streaming rather than index random access
- based on the scatter-gather programming model
- processing in-memory and out-of-core graphs on a single-shared memory machine

Roy, Amitabha, Ivo Mihailovic, and Willy Zwaenepoel. "X-Stream : edge-centric graph processing using streaming partitions." Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, 2013.

State of the art {Data Flow Graphs }

Naiad : A Timely Dataflow System

Timely dataflow model

- Timely dataflow model
  - Computational model based on a directed graph
  - Each message is labeled with timestamp : recognition of data input epochs + loop iterations
- Naiad : A prototype distributed implementation of timely dataflow model
  - A Naiad cluster : Number of processes hosting workers
    - Data exchange betweeen workers : Shared Memory
    - Message exchange between processes : TCP connections

Murray, Derek G., et al. "Naiad : a timely dataflow system." Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, 2013.

Questions or Suggestions ?