

A Portable and Efficient Thread Library for OpenMP

Sven Karlsson

*Department of Microelectronics and information Technology
Royal Institute of Technology, KTH, Sweden
email: sven.karlsson@imit.kth.se*

Abstract

The design of a portable, yet efficient, thread library, called Balder Threads, is discussed in this paper. The library is used within Balder, a run-time library for OpenMP 2.0. The thread library is evaluated using the EPCC micro-benchmarks and measuring the overheads for the entire Balder OpenMP run-time library. The overheads, using Balder Threads, are found to be an order of a magnitude smaller than when using POSIX primitives. The overheads achieved are comparable to those of a commercial system from Intel.

1. Introduction

A compilation system for OpenMP typically consists of an OpenMP aware compiler and a run-time library system. The run-time library not only provides the intrinsic OpenMP functions as described by the OpenMP specification but also generally provides important support functions used by the compiler for thread creation and thread synchronization. The efficiency of these support functions directly influences the performance of OpenMP constructs.

One OpenMP run-time library is the *Balder* library which is targeted by the OdinMP compiler [3, 4, 5]. The library and the compiler fully support OpenMP 2.0 including nested parallelism [6]. The Balder library has a modular design and has several sub-libraries. One such sub-library is an efficient portable thread library called *Balder Threads*. In this paper the functionality of the primitives in Balder Threads is discussed as well as the design of the library. It is shown, using the EPCC micro-benchmarks [1], that the Balder Threads library provides an order of magnitude better performance than POSIX primitives for a number of common OpenMP constructs while still being highly portable.

The existing non-commercial run-time libraries either use specialized non-portable thread libraries or POSIX primitives which incur a high overhead. The, in this paper, proposed system provides portability while still being efficient.

The rest of the paper is organized as follows. The Balder and Balder Threads libraries are discussed in section 2. Experimental results are presented in section 3 while the paper is concluded in section 4.

2. Balder and Balder Threads

Balder is an OpenMP run-time library supporting OpenMP 2.0 [4]. The library supports SMPs as well as clusters of SMPs and is designed as a layered system, see figure 1.

The library consists of three sub-libraries. The sub-libraries are *Balder Messages* which is a messaging library used when targeting a cluster, *Balder Oslib* which provide an operation system independent interface to operating system resources, and finally Balder Threads which provides thread primitives. The actual OpenMP functionality is implemented in a layer which is built on-top of the mentioned three libraries. The OpenMP layer also includes a software distribution shared memory system, software DSM system, which provides a shared address space when targeting a cluster. The software DSM system, as well as the messaging layer, is completely inactive when using a single SMP node and will not inflict any overhead in that case. I will not touch on clustering issues in this paper.

The entire run-time library is written in C using standardized operating system APIs to achieve portability. Some operation system services needed by the library, such as the virtual memory system, do not have standardized APIs but the Balder Oslib provides a virtualization layer to these services.

Balder has been ported to several different variants of Unix as well as various revisions of Windows. It has also been ported to several processor architectures.

The focus of this paper is the Balder Threads library which provides threading primitives. In short the following primitives are provided:

- Thread creation and thread destruction.
- Thread synchronization through monitors.
- Barriers.
- Work-queues.

Internally, the OpenMP layer manages a pool of threads. This means that threads are only created when the thread pool is empty and so thread creation need not be very efficient. The other primitives need however be very efficient. The thread synchronization primitives are for instance used internally by Balder for mutual exclusion and also for the intrinsic OpenMP lock functions and the critical construct.

The Balder Threads library uses the POSIX thread API for thread creation, thread destruction and synchronization [2]. The POSIX thread API is a general thread API and has, as can be seen in section 3, a fairly high overhead. To improve on the overheads, Balder Threads implements a layer, on-top of functions from the POSIX thread API, which performs synchronization using processor architecture dependant primitives such as test-and-set and fetch-and-add. Functions from the POSIX thread API are then used to block threads. The processor dependant primitives are implemented as short assembly language snippets embedded in pre-processor macros making it possible to implement the synchronization primitives in portable C code. Only the processor dependant macros need to be implemented to port Balder Threads to a new processor architecture variant.

The basic form of synchronization in the Balder Threads library is spin-locks based on test-test-and-set. The target for the Balder library is SMPs up to roughly 8 CPUs and so more elaborate synchronization primitives was deemed not needed. Experiments conducted shows that the test-test-and-set procedure used is reasonable even up to as much as 64 CPUs.

As mentioned, the Balder Threads library uses a number of macros to make use of processor dependant primitives. The set of primitives supported through the macros are:

- Test-and-set. This is, as outlined above, the basic synchronization primitive and is the only synchronization primitive which is required by the library to be implemented in a port.
- Memory fences. The library makes use of memory fences if needed by the underlying processor architecture. The library assumes that the memory fences control both read and write operations and inserts fences before reading, and after writing, shared variables used for synchronization. The memory fence macros are blank for architectures whose memory consistency models do not require memory fences.
- Fetch-and-add. The fetch-and-add is used through out the library to reduce synchronization cost for various shared variables that are used as counters. The primitive is in particular used to reduce the overhead of barriers and work-queues.

In addition to these macros there are macros, required by the library, which define the layout of stack frames and define how functions are called. These macros are used to implement the parallel construct. Required macros also define the size and layout of cache lines. These macros are used to lay out spin-lock variables so that each spin-lock resides in its own cache

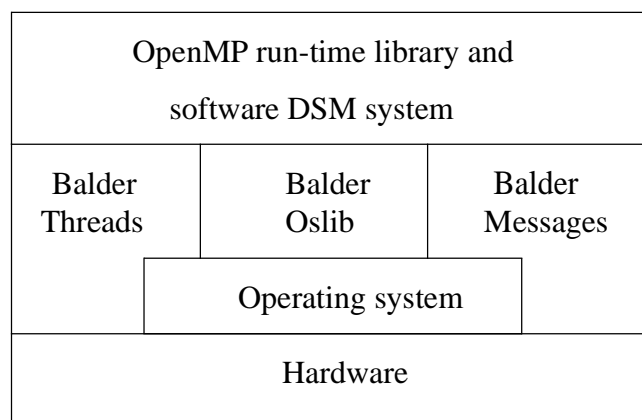


Figure 1. Overview of the design of the Balder library.

Table 1. Overheads in microseconds of common OpenMP constructs.

OpenMP Construct	Intel compiler	Balder with OdinMP without architecture support	Balder with OdinMP and architecture support
Parallel construct	1.43 +/- 0.11	27.15 +/- 1.35	2.91 +/- 0.15
For construct	0.79 +/- 0.17	10.59 +/- 0.87	2.93 +/- 0.30
Barrier construct	0.48 +/- 0.19	11.14 +/- 0.29	0.49 +/- 0.12
Lock and unlock primitives	0.48 +/- 0.33	6.90 +/- 0.35	0.47 +/- 0.12

line so as to make sure coherency traffic is kept at a minimum. POSIX primitives are used as efficiently as possible for synchronization if no macros are provided.

The test-test-and-set variant used in Balder Threads performs a time-out if a spin-lock has not been released within certain time period of busy-waiting. The actual time-out period is set to a time roughly equal to two context switches at system startup through run-time measurements on POSIX thread API primitives but the time-out period can also be fine-tuned through command-line options. The purpose for the time-out period is to avoid excessive busy-waiting in systems where several processes compete for processor resources.

A pseudo code describing the test-test-and-set procedure and the monitor entry and leave functions is outlined in figure 2.

3. Experimental results

A few experiments have been conducted so as to assess the performance of the thread library. A dual Pentium-III workstation running Linux version 2.4.25 was used as experimental platform. The processors were running at a clock rate of 1 GHz.

The EPCC micro-benchmark suite was used in the experiments [1]. The benchmarks were compiled with OdinMP version 0.284.1 and GCC 3.3.4. The Balder library version 0.105.1 was used and was also compiled with GCC 3.3.4.

For comparison, the same set of benchmarks were compiled with the Intel C/C++ compiler version 8.0 and run on the experimental platform. The Intel compiler supports OpenMP. The Balder library cannot currently be compiled with the Intel compiler. The highest available optimization level was used in both compilation systems.

The overheads in microseconds for the most common OpenMP constructs are summarized in table 1. The overheads are presented with their 95% confidence interval. The overheads in the first three rows are for one single parallel region, parallel for-loop, and barrier respectively. The lock and unlock primitives row is the overhead of setting and then releasing a single lock once. Two versions of the Balder library were used. One with full architecture support including test-and-set and fetch-and-add operations and one where all architecture support is disabled forcing POSIX primitives to be used for all synchronization.

The overheads are reduced by as much as an order of magnitude when using architecture support when compared to pure POSIX primitives. The overheads for synchronization primitives are as good as for the commercial Intel compiler while higher for parallel for-loops and creation of parallel regions. The reason lies in the way the OdinMP compiler generates code and how parallel regions are handled by the run-time library.

4. Conclusions

In this paper, the thread library in the Balder OpenMP run-time is presented. The thread library is called Balder Threads and is highly portable. The paper discusses the design of Balder in general and Balder Threads in particular. The performance of the thread primitives are evaluated using the EPCC micro-benchmark suite and is found to be an order of magnitude better than that of POSIX primitives. The performance is also found to be comparable to that of a commercial compilation system from Intel.

```

int test_test_and_set(spin_lock_type* spin_lock_variable)
{
    do
    {
        MFENCE;          /* MFENCE is a macro defining
                           a memory fence operation. */
        while(*spin_lock_variable==TAKEN)
        {
            MFENCE;      /* Wait and perform memory fences
                           until the spin-lock variable is
                           free. */

            if (timeout occurred)
            {
                return 1;
            }
        }
    } while (TEST_AND_SET(spin_lock_variable)==TAKEN);
    /* TEST_AND_SET is a macro defining
       a test_and_set primitive. */

    return 0;
}

void enter_monitor(monitor_variable_type* monitor)
{
    while (test_test_and_set(&monitor->spin_lock_variable))
    {
        /* A time-out has occurred. The
           thread will be blocked using
           POSIX functions and condition
           variables. */

        block thread;

    }
}

void leave_monitor(monitor_variable_type* monitor)
{
    monitor->spin_lock_variable=FREE;
    MFENCE;
    if (threads are blocked)
    {
        unblock at least one thread;
    }
}

```

Figure 2. Pseudo code for monitor entry and monitor leave.

Acknowledgements

The work in this paper has been partly financed by the European Commission in the Intone project under contract number IST-1999-20252.

References

- [1] J. M. Bull, Measuring Synchronization and Scheduling Overheads in OpenMP, in *Proceedings of the First European Workshop on OpenMP*, Sept. 1999, pp. 99-105
- [2] IEEE, *IEEE std 1003.1-1996 POSIX part 1: System Application Programming Interface*, 1996
- [3] S. Karlsson and Mats Brorsson, A Free OpenMP Compiler and Run-Time Library Infrastructure for Research on Shared Memory Parallel Computing, to appear in *proceedings of The 16th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2004)*, November 2004
- [4] S. Karlsson, *Balder – An OpenMP run-time library for clusters of SMPs*, Technical report, TRITA-IMIT/LECS R 04:01, ISSN 1651-4661, ISRN KTH/IMIT/LECS/R-04/01--SE, Department of Microelectronics and Information Technology, Royal Institute of Technology, KTH, August 2004
- [5] S. Karlsson, *OdinMP homepage*, <http://www.odinmp.com>, retrieved on August 8th 2004
- [6] OpenMP consortium, *OpenMP C and C++ Application Program Interface, Version 2.0*, 2002