# Optimized Linux Code for Intel XScale® Microarchitecture

## Application Note

***July 2005***

Order Number: 308517001US
June 2005

# Contents



## Figures



## Tables

# Revision History

| Date | Revision | Description |
|---|---|---|
| June 2005 | 001 | Initial Release. |

# 1.0 Purpose

This document outlines recommendations to be made to the software architecture of a software product running on I/O processors (IOPs) using Intel XScale® microarchitecture.[1] The objective of the recommendations is to optimize the overall system performance by making enhancements to the software architecture used with external storage products.

# 2.0 Related Documents

- *Data Access Performance Optimization on the Intel® 80321 I/O Processor White Paper* (http://developer.intel.com/design/iio/papers/273872.htm)

# 3.0 Background

As a result of the increased usage of Linux* on IOPs using Intel XScale® microarchitecture, Intel has identified several recommendations to the software architecture that can be made to improve the overall system performance. Intel has seen that the more of these recommendations that are implemented, the better the system-level performance.

# 4.0 Kernel Mode to User Space Transitions

Linux operates with a separate user and kernel space. Each time data is passed between user and kernel space a data copy occurs. These data copies cause a performance hit each time they occur. To minimize this impact, Intel has identified and developed specialized copy routines that utilize the DMA engine. This information can be found in the Intel XScale® microarchitecture IOP Linux patches located at sourceforge.net/projects/xscaleiop/.

The DMA engine is utilized for data copies of 4 K blocks and above (this value can be adjusted based on the customer applications). Intel has found that 4 K block threshold for a generic Linux kernel to be optimal. Additionally, Intel recommends minimizing the number of user-to-kernel and kernel-to-user transitions. The most optimal mode of operation is to operate exclusively out of the Kernel space. There is potential for up to three times the improvement in performance, operating exclusively out of Kernel space. Using compile time wrappers to substitute user and kernel calls enables the code to be moved easily between user and kernel space, thereby allowing easy debug in user space and along with high performance in kernel space.

# 5.0 Context Switching

Linux also utilizes multiple working processes. Each time that the schedule chooses a new process, a context switch is generated. Each context switch causes the data cache to be flushed and the instruction cache to be invalidated. The higher the frequency of context switches, the more of an impact there is on system performance. Intel strongly recommends minimizing the number of active working processes to reduce the number context switches.

1. ARM* architecture compliant.

# 6.0 Optimization Level

The latest compiler Intel is recommending is the GNU gcc version 3.4.2. Make sure that the -O2 optimization flag is specified. Sometimes moving to an -O3 optimization flag can further increase performance, but not in all cases.

# 7.0 Code Bloat

By performing a thorough code analysis, it is possible to identify opportunities to reduce the amount of code that exists in some routines (code bloat). One way to do this is to conditionally compile out unnecessary code such as debug code or sanity checking code and inefficient loops. The improvements made by this process result in better performance and greater robustness of the code.

# 8.0 Instruction Cache Usage

Through a careful analysis of the critical performance path, areas of coalescing can be identified. The objective is to have each section of code process as much information / commands as it possibly can before passing the data onto the next code module. This results in a more efficient use of the Instruction Cache, by allowing more work to be done by the code that is located in the Instruction Cache. The trade-off is the added latency for some data blocks, but the improved throughput reduces the average overall latency for all blocks. Classically this is one of the reasons why major blocks are constructed in separate threads.

# 9.0 Additional Suggestions

The reference Whitepaper *Data Access Performance Optimization on the Intel® 80321 I/O Processor White Paper* contains additional lower level recommendations, along with some code examples.