The Journal of Systems and Software 98 (2014) 25–43

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

DRE system performance optimization with the SMACK cache efficiency metric

Hamilton Turner[a,*], Brian Dougherty[b], Jules White[b], Russell Kegley[c,1], Jonathan Preston[c,1], Douglas C. Schmidt[b], Aniruddha Gokhale[b]

[a] Virginia Polytechnic, United States
[b] Vanderbilt University, United States
[c] Lockheed Martin Aeronautics, United States

ARTICLE INFO

Article history:
Received 2 December 2013
Received in revised form 26 June 2014
Accepted 16 August 2014
Available online 27 August 2014

Keywords:
DRE
Deployment
Optimization
Heuristic
Cache

ABSTRACT

System performance improvements are critical for the resource-limited environment of multiple integrated applications executing inside a single distributed real-time and embedded (DRE) system, such as integrated avionics platform or vehtronics systems. While processor caches can effectively reduce execution time there are several factors, such as cache size, system data sharing, and task execution schedule, which make it hard to quantify, predict, and optimize the cache usage of a DRE system. This article presents SMACK, a novel heuristic for estimating the hardware cache usage of a DRE system, and describes a method of varying the runtime behavior of DRE system software without (1) requiring extensive safety recertification or (2) violating the real-time scheduling deadlines. By using SMACK as a maximization target, we were able to reduce integrated DRE system execution time by an average of 2.4% and a maximum of 4.34%.

1. Introduction

Current trends and challenges: Distributed real-time and embedded (DRE) systems, such as integrated avionics systems and vehtronics systems, are subject to stringent real-time constraints. To ensure these real-time requirements are met, these systems must minimize software execution time. One approach to reduce DRE execution time is to reduce the time spent loading data from memory by efficiently utilizing processor caching hardware.

Multiple design techniques have been researched for reducing system execution time by increasing processor cache utilization. For example, Bahar et al. (2005) examined several different cache techniques for reducing execution time by increasing cache utilization efficiency. Their experiments showed that efficiently utilizing a processor cache can result in as much as a 24% reduction in execution time. Likewise, Manjikian and Abdelrahman (1995) demonstrated a 25% reduction in execution time as a result of modifying the source-code of the executing software to use cache partitioning.

Many optimization techniques (Reineke et al., 2007; Nayfeh and Olukotun, 1994; Sprangle et al., 2002) exist to increase how efficiently caches are utilized by modifying application source code to increase the *temporal locality* of data accesses, which defines the proximity with which shared data is accessed in terms of time (Kowarschik et al., 2003). For example, loop interchange and loop fusion techniques can be used to increase temporal locality of accessed data by modifying software application source code to change the order in which application data is written to and read from a processor cache (Kowarschik et al., 2003; Manjikian and Abdelrahman, 1995). Increasing temporal locality increases the probability that data common to multiple tasks will persist in the cache, resulting in reduced cache-misses and software execution time (Kowarschik et al., 2003; Manjikian and Abdelrahman, 1995).

In general, however, prior work has focused on source-code level modifications for single applications, which is problematic for DRE systems built from multiple integrated applications, such as the architecture shown in Fig. 1. Source code modifications in an integrated DRE system are infeasible due to 1) the proprietary nature of individual applications, and 2) safety requirements which necessitate extremely extensive certification after any source code modifications.

* Corresponding author. Tel.: +1 6158183577.
E-mail addresses: hamiltont@vt.edu, hamiltont@gmail.com, hturner0@vt.edu (H. Turner), briand@dre.vanderbilt.edu (B. Dougherty), julesw@dre.vanderbilt.edu (J. White), russell.b.kegley@lmco.com (R. Kegley), jonathan.d.preston@lmco.com (J. Preston), schmidt@dre.vanderbilt.edu (D.C. Schmidt), gokhale@dre.vanderbilt.edu (A. Gokhale).

[1] This work was sponsored in part by the Air Force Research Lab.

http://dx.doi.org/10.1016/j.jss.2014.08.031
0164-1212/

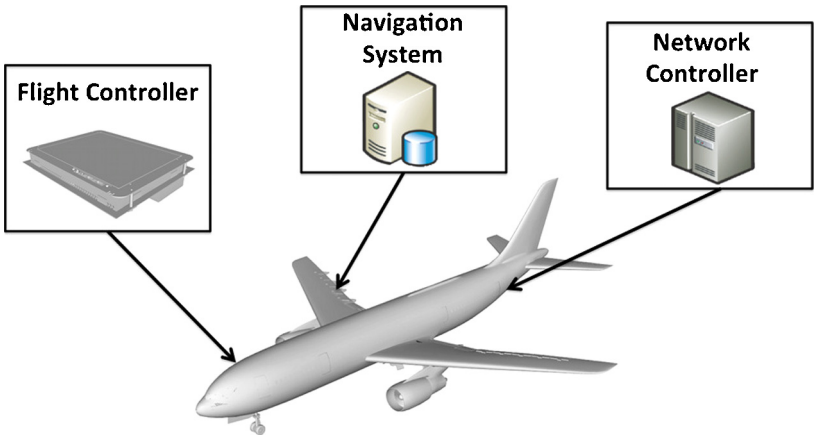

Fig. 1. Example of an integrated avionics system.

An integrated DRE system is composed of many individual applications, which are frequently provided by distinct subcontractors. License issues may prevent access to the source code, or prevent the right to modify or recompile the source code. Moreover, DRE system designers may not have the expertise to safely modify application-specific source code to improve the temporal locality of the source code.

Integrated DRE systems are often subject to stringent safety requirements, such as bounding the frequency at which hardware failures can occur and preventing code-level errors in application software, middleware, and operating systems. To help ensure predictable behavior, applications in integrated DRE systems must undergo a rigorous safety inspection process. After this process is completed and the result has been certified, any alteration to an application may invalidate the certification. Safety requirements thus pose a significant barrier to the use of cache optimization techniques that require source code alterations.

Solution approach → Heuristic-Driven Schedule Alteration of Same-rate Tasks to Increase Cache Utilization.

Priority-based scheduling techniques can help ensure DRE system software executes without missing real-time deadlines. For example, rate-monotonic scheduling (Pingali et al., 2007) is a technique for creating task execution schedules that satisfy real-time constraints by assigning priorities to tasks based on the task periodicity and ensuring utilization bounds are not exceeded. These tasks are then split into sets that contain tasks of the same priority/rate.

Rate-monotonic scheduling specifies that tasks of the same rate can be scheduled arbitrarily (Dhall and Liu, 1978). Fig. 2 shows two different valid task execution schedules generated with rate-monotonic scheduling. As Task A1 and Task B1 share the same priority, their execution order can be swapped without violating real-time constraints. Each scheduling problem with at least one valid execution order therefore has a number of equally valid permutations, which can be created by rearranging the order of same-rate tasks. This research shows that it is possible to improve the cache hit-rate of integrated DRE systems by selecting a valid execution order that increases the temporal locality of data accesses.

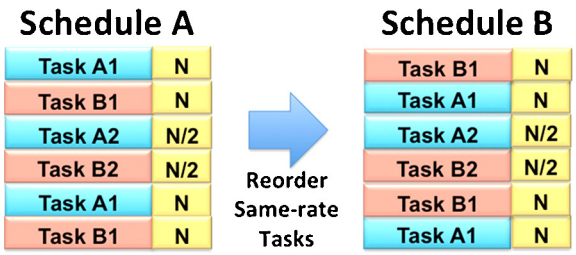

Fig. 2. Valid task execution schedules.

Critically, our approach for improving cache utilization of integrated DRE systems does not require any source code modifications. This allows DRE system integrators to improve integrated DRE system performance without invalidating application safety recertification, requiring legal agreements to share source code, or needing specialized expertise for each application. Our modifications only change the execution order of same-rate tasks, which results in a system that is still a valid rate-monotonically scheduled application as interchanging tasks of identical utilization does not change the value of the Liu–Layland bound (Dhall and Liu, 1978). To select the best valid schedule from the set of all possible valid schedules, we introduce the *System Metric for Application Cache Knowledge* (SMACK) heuristic, which measures the maximum possible cache utilization of a given execution schedule. SMACK considers several factors, such as cache size, data sharing, and task execution schedule, to provide developers with a way to determine which orderings of same-rate tasks have higher potential for cache utilization. SMACK enables DRE system designers to manipulate models of system runtime behavior without having to repeatedly construct and measure potential system implementations, thereby yielding performance increases without expending undue effort on multiple implementations.

This article provides the following contributions to integrated DRE system creation and deployment:

- We present a real-time scheduling heuristic for same-rate tasks that satisfies real-time scheduling constraints and safety requirements, increases cache hits, and requires no new hardware, software, or middleware. Proper use of this heuristic enables reduction in execution time of rate-monotonically scheduled DRE systems without requiring recertification.
- To motivate the need for scheduling enhancements to improve the efficiency of cache utilization in integrated DRE systems, we present an industry case study of an integrated avionics system in which modifications to the constituent applications are prohibitively expensive due to safety certification requirements.
- We provide a formal methodology for calculating the "SMACK score", which quantifies the temporal locality of different execution schedules for the same-rate tasks in an integrated DRE system.
- We present empirical results of the performance of 44 simulated integrated DRE systems, each with different data sharing characteristics and task execution schedules, and demonstrate that the calculated SMACK score correlates with an increased cache