International Journal of Cyber-Physical Systems
Volume 1 • Issue 1 • January-June 2019

Efficient Dynamic Memory Management for Multiprocessor Cyber-Physical Systems

Ali Ahmadinia, California State University, San Marcos, USA

ABSTRACT

Dynamic data management for multiprocessor systems in the absence of an operating system (OS) is a challenging area of research. OSs are typically used to abstract developers from the process of managing dynamic data at runtime. However, due to the many different types of multiprocessor available, an OS is not always available, making the management of dynamic data a difficult task. In this article, we present a hardware and software co-design methodology for the management of dynamic data in multiprocessor system on chips (MPSoC) development environments without an OS. We compare and contrast the method of sharing dynamic data between cores with standard methods and also to static data management methods and find that the proposed methodology can improve the performance of dynamic memory operations by up to 72.94% with negligible power and resource consumption.

KEYWORDS

Memory Management, Multiprocessor Architectures, Operating Systems, Power

INTRODUCTION

Multiprocessor system-on-chips (MPSoCs) are popular due to their suitability for multithreaded applications. MPSoCs are characterized by the number of processors (cores) they possess; the bus interconnects between cores and main memory, and the types of memories available. This work focuses on MPSoCs that do not have access to an operating system (OS), as this is an interesting and challenging area of memory management that is applicable to both academia and industry. Due to the wide range of MPSoC attributes and platforms available, there is no known OS that can be used to manage applications and memory for every MPSoC configuration. Furthermore, the presence of an OS does not always improve the development of applications for MPSoCs (Wolf, Jerraya, & Martin, 2008) and the choice of whether or not to use one is dependent on the application, development time, and resources.

For these reasons, we focus our attention on dynamic memory management (DMM) techniques that can be used in MPSoCs where cores operate independently w.r.t each other, and no OS is available. We compare and contrast the use of dynamic and static data placement methods and present a hardware/ software co-design methodology that allows independent cores to view and access the dynamic allocations of other cores - a process that would not be possible through standard DMM methods alone. We demonstrate how private heaps can become globally shared heaps in the MPSoC environment and how DMM can be easily implemented using light-weight software protocols and hardware-based directories.

DOI: 10.4018/IJCPS.2019010103

We test our proposed solutions using a data-intensive face detection algorithm allowing a thorough evaluation of how data placement and management techniques can affect face detection applications, and evaluate performance based on the improvement of execution time they harness. The remainder of this paper is organized as follows. The next section II gives an overview of the relevant literature and distinguishes our work from it. The third section formulates the DMM problem, and afterward, the proposed hardware/software co-design methodology is described. Then the results from our tests, as well as a comparison to the literature are presented. We conclude with final remarks and future work in the last section.

RELATED WORK

Anagnostopoulos et al. (Anagnostopoulos et al., 2011) perform microcode optimizations for the DMM of an MPSoC through a hardware-based dual-microcoded controller. A dynamic multithreaded application is statically analyzed to obtain DMM decision trees, where exploration is performed based on a constraint-orthogonal partition methodology, similar to (Atienza, Mendias, Mamagkakis, Soudris, & Catthoor, 2006; Xydis, Bartzas, Anagnostopoulos, Soudris, & Pekmestzi, 2010). Variable sized and types of heaps, with local or global visibility (w.r.t other cores), can be refined for different MPSoC topologies. However, allocations made to local or global heaps are at most 1500 bytes in size and are stored in SDRAM. Therefore, in order for an application to manage larger dynamic allocations, a large amount of SDRAMs would be consumed.

Atienza et al. (Atienza et al., 2006) present a methodology for the management of memory and the reduction of the footprint of dynamic memory allocations for embedded multimedia and wireless applications. Fragmentation was found to impact the memory footprint of systems and was reduced by applying their memory management methodology, albeit with an increase in execution time.

Koutras, Bartzas, and Soudris (Koutras, Bartzas, & Soudris, 2012) investigate the use of adaptive dynamic memory allocators to predict the amount of memory an application might want. Runtime information is used to ascertain the current memory requirements of the application and the current state of memory space. A goodness factor is used to predict future memory accesses and how blocks are allocated.

Xydis et al. (Xydis et al., 2010) define a design space for multithreaded DMM for MPSoCs and explore it to find the optimal solution. They extend a previous single-threaded DMM design space model (Atienza et al., 2006) and capture multiprocessor and multithreaded decisions. However, the work assumes the presence of an OS to manage heaps for threads and the MPSoC which is quoted as being an Intel Quad Core MPSoC operating at 2.66 GHz, making it uncertain of the applicability to embedded systems.

The platforms used in the above work are a mixture of network on chips (NoCs) (Anagnostopoulos et al., 2011), commercial desktop processors not suitable for embedded systems (Koutras et al., 2012; Xydis et al., 2010), and unknown simulators (Atienza et al., 2006). In this work, we focus on MPSoCs implemented on field programmable gate arrays (FPGAs) with soft-core processors - processors that only exist on logic fabric when explicitly instantiated. Our DMM scheme only focuses on dynamic memory operations to heaps that reside in main memory for the time being. Our proposed scheme differs from the focus of (Bathen, Dutt, Shin, & Lim, 2011), where they aim to provide dynamic *SPM* memory allocation support.

PROBLEM FORMULATION

The problem of soft-core MPSoCs with no global OS can be summarized by the following three points:

1. No OS is available to manage/unify cores together.