VNU Journal of Science, Mathematics - Physics 23 (2007) 15-21

Dynamically reconfiguration architecture for embedded systems using Kaffe

Vu Quang Dung*, Nguyen Viet Ha, To Van Khanh
Department of Software Engineering, College of Technology, VNU
144 Xuan Thuy, Cau Giay, Hanoi, Vietnam

Received 28 May 2007

Abstract. In designing embedded systems, the exploration and synthesis of different design alternatives and co-verification of a specific implementation are the most demanding tasks. Kaffe, an open-source technology, provides a platform for building a runtime environment and integrating different design methodologies. Integrating Kaffe into embedded systems is the cornerstone of java-like technologies, allowing possibilities such as the development of portable programs on mobile devices. In this paper, we introduce a co-design environment based on Kaffe which supports the specification and prototyping of dynamically reconfigurable embedded systems.

Keywords: Embedded systems, Kaffe Virtual Machine (KVM), real-time systems, ARM, Hardware-Software co-design, Java Native Interface, Kaffe Native Interface

1. Introduction

Nowadays, embedded systems are playing a major role on the development of technology. Embedded systems are present in almost all mobile and electrical devices. The goals of embedded systems are long system lifetime, low cost, low development time, and portability.

Some mobile devices such as PDA and cellular phones often have an environment using the Java virtual machine [1]. But these Java environments are not optimized for supporting special embedded programs: they cannot be modified nor controlled, and they are very expensive. Therefore, in the design of embedded systems executing java bytecode, we have developed a design exploration and prototypying platform, using an open source Kaffe technology for embedded platform [2]. The embedded Kaffe technology allows the development of efficient and secure cross-platform software, and the Kaffe virtual machine (KVM) is the cornerstone of this technology.

2. Specification of eEmbedded Kaffe systems

The target architecture for such systems consists of a microprocessor running a Kaffe virtual machine, and a hardware processor consisting of a core from the ARM family. At a high level, Kaffe is a system that integrates C/C++ and Java for the native environment. An overview of a Kaffe architecture is illustrated in Figure 1.

* Corresponding author. Tel: 84-4-7549016
E-mail: Dungvq@vnu.edu.vn

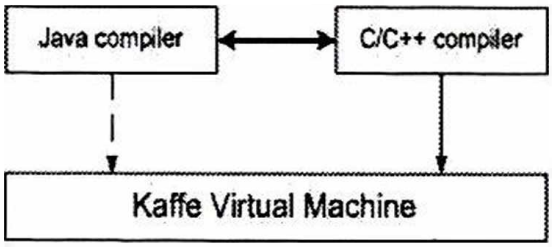

Figure 1. Overview of Kaffe architecture.

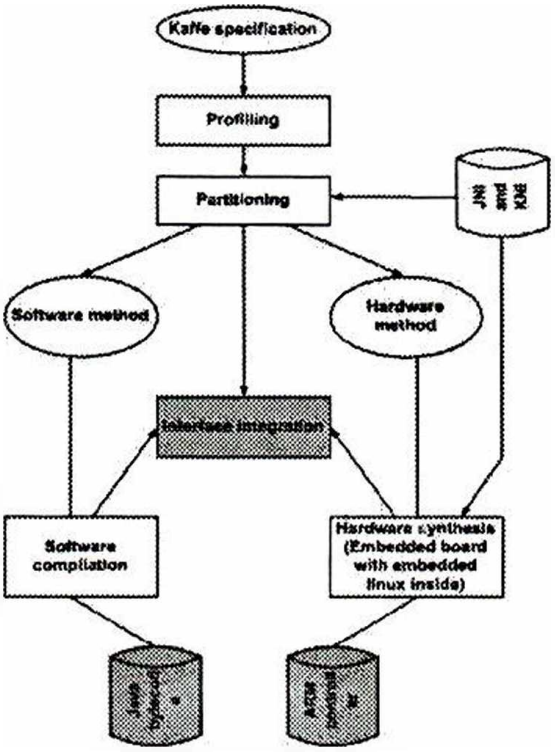

Figure 2. Co-design of Embedded Kaffe.

For the purpose of using a java-like technology for embedded systems, we design a distributed Kaffe system using the Kaffe virtual machine to execute the Java bytecode. The KVM relies on an interpretation mechanism which emulates the execution of Java bytecode on mobile devices based on embedded systems. The system architecture consists of a microprocessor running a Kaffe environment and porting to hardware controlled by an ARM processor. Starting from an initial Kaffe specification, profiling data is gathered while executing the program with input data. The profiling data is then visualized to guide the designer in the partitioning process. Partitioning is done at the method level of granularity using a user interface. Functions which are to be implemented in hardware are synthesized using high-level logic synthesis tools. Java bytecode is stored in the pool of software methods. For all methods which are candidates for the implementation in reconfigurable hardware, the ARM processor controls all information storing in the pool of hardware methods. The target software platform for porting Kaffe environment is an embedded Linux, adapted to real-time systems. A co-design environment using Kaffe is showed in Figure 2.

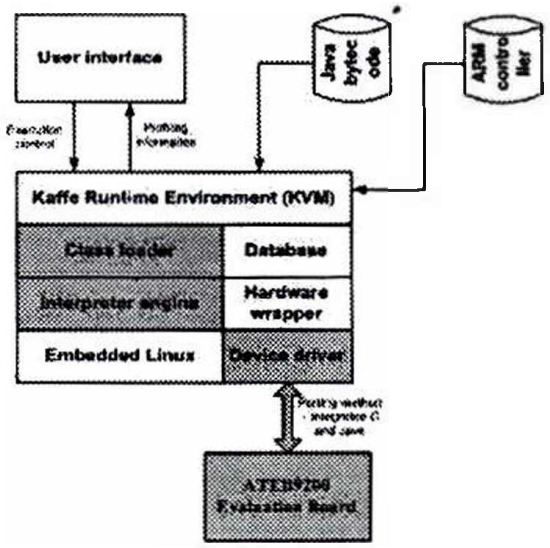

Figure 3. Integrated platform.

3. Co-processor hardware design based on an ARM processor

To manage system integration, a runtime management program has been implemented. The runtime management program schedules methods for execution either as software on the KVM or as hardware on the ATEB9200 Evaluation board with an ARM processor controller. Execution on this system is a dynamic process [3]. The execution flow of the system is dominated by the software part, executed on the KVM. The core component of the runtime environment is the KVM. It consists of a class loader for dynamically loading Java bytecode and an execution engine for interpreting these bytecode on the ARM processor. An integrated system is shown in Figure 3.

The class loader is extended for reading in the current system partitioning table and for handling hardware methods, executed on the ATEB9200 Evaluation board. The execution engine needs to know whether a method is to be interpreted as bytecode or executed in the ARM processor. For this reason, the class loader assigns a special flag to every hardware method. During execution of the application, the interpreter has to activate the hardware call module whenever the flow of control reaches a hardware method. The hardware wrapper implements the procedures, which transfers input data to the board, executes on the ARM processor and transfers the resulting data back to the calling thread.

For the ease of using the system model above, we propose some ideas for a simple implementation of Kaffe in the co-processor hardware on an ATEB9200 board that will lead to an increase in performance [4], as shown in Figure 4. Within the block diagram, all connections are 32 bits wide with one exception. This is due to the fact that Java byte-code is built on a 32-bit architecture. The exception is the connection between the Stack cache and the arithmetic units that is 96 bits. This allows for long operands to pass from the cache to the arithmetic units in a single cycle.

The logical blocks that are unique to this design are the ARM I/O interface and the interface controller. The interface is connected inside the ATEB9200 board to both the instruction and data caches as well as the interface controller. Instructions for execution flow through the interface and into the instruction cache for execution.

The interface controller maintains the link between the Kaffe system and the ATEB9200 board. It is responsible for halting the hardware in the event that execution needs to stop while the KVM carries out part of the execution. In addition, it is responsible for changing the context of the current

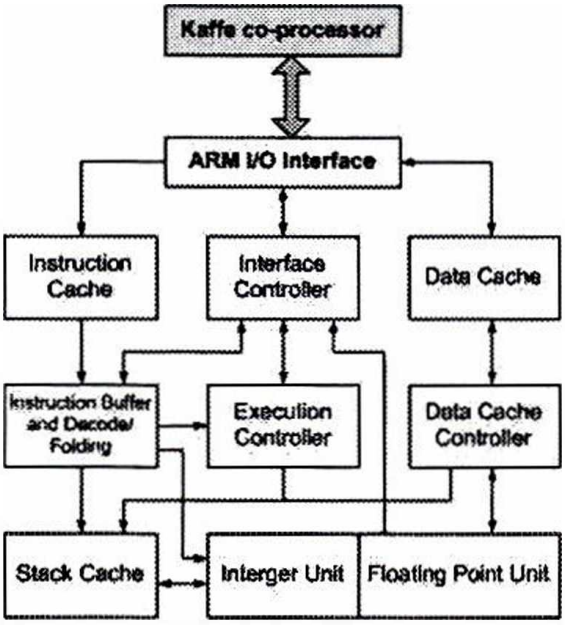

Figure 4. Hardware design based on an ARM processor.

execution when signaled by the KVM. In essence, it is the hardware mediator between the hardware and software.

4. Software co-design

Between the Java co-processor, which is in the ARM processor, and the memory that is available on the board, there must be some interaction. The memory on board is used as an intermediate memory location between the ATEB9200 system and the Kaffe co-processor. As the ATEB9200 system resolves classes and sets the location of execution with the Kaffe coprocessor, it places into the board's dynamic memory RAM the class itself and the instructions that are located at the address of execution. The Kaffe co-processor through its I/O interface retrieves data from the memory and brings it into the instruction cache on the ATEB board for execution, similar to [5]. The memory must also contain any data that cannot fit into the data cache and is used too often to be swapped back out onto the ATEB system.

To handle the amounts of data that are being transferred back and forth between the ATEB9200 system and the card, in the event that the memory on board is not sufficient to hold all the application and necessary data generated, the memory must be split into manageable blocks to allow for quick and efficient transferring.

For this reason and based on [6], we propose in Figure 5 the software design of Kaffe co-processor, which supports the execution of the Java byte-code on the ATEB9200 board. This design consists of the threads of execution that are necessary in the software to support the Kaffe co-processor. Each of the threads are standard threads within the Kaffe virtual Machine except for the Hardware handler that is added for communicating with the ARM processor interface. The hardware handler is also an essential component in the communication of data between the Kaffe co-processor and the Java byte-code. The thread is also used to maintain the synchronization of data between the memory on the ARM processor and the memory located on the ATEB9200 board.

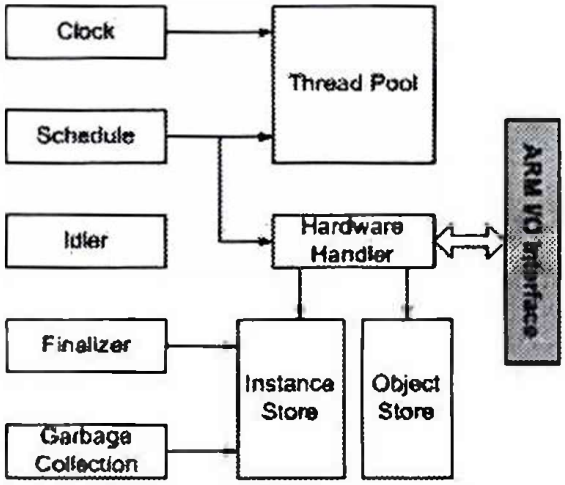

Figure 5. Kaffe design of co-processor.

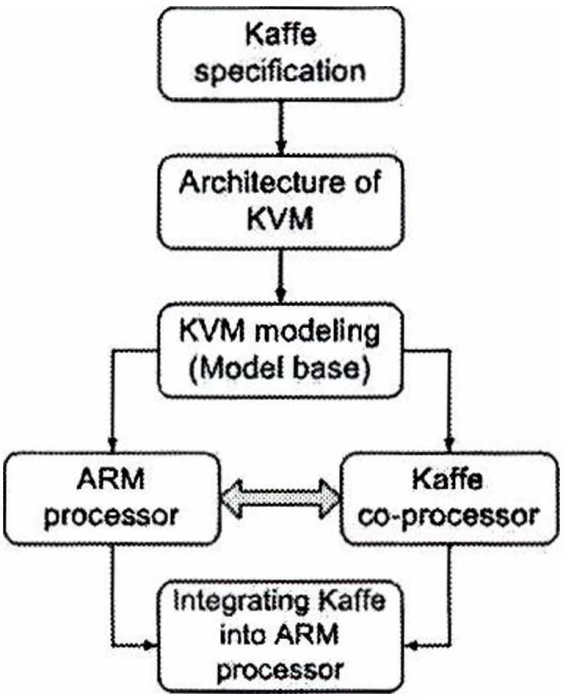

Figure 6. ARM/KVM integration.

5. An implementation of Kaffe architecture based on an ARM processor

Based on [6] and with virtual machine Java-like system design in [4], we have ported Kaffe into an Embedded system based on ARM processor [7]. We implemented a new model of hardware and software integration, which executes Java byte code on a hardware device based on the ARM processor, as shown in Figure 6.

We called a structure of the ARM processor on a Kaffe system following config/arm/jit.h. When the class is loaded, we built a dispatch table, which contains the pointers to methods. Since the methods are not yet translated, each dispatch table entry points to a trampoline for the associated method as written in [8], that has the structure

```
typedef struct _methodTrampoline {
```

```
    unsigned char call arm_PACKED;
    int fixup arm_PACKED;
    struct _methods* meth arm_PACKED;
}methodTrampoline;
```

This structure calls to a native method of the Kaffe Native Interface (KNI) and builds in standard C using a call for information, which will be stored in the ARM dynamic memory. The algorithm of this method forms the major function calls in Kaffe's main.c, that provides an overview of the entire compilation process of Kaffe.

ALGORITHM for Kaffe on the ATEB9200 device based on ARM processor [9]:
BEGIN

- call machine-specific main.
- set java_version.
- get default VM init args.
- setup libtool, device memory.
- get the CLASSPATH, LIBRARYPATH, KAFFEHOME environment variables.
- process the program options.
- get the class name to start with
  - * create and initialize the Java VM (calling JNI based on KNI).
  - * find the class to start with (JNIEnv *env, char *argv[], int farg, int argc).
  - * get the method handle for the main method in the class to start with (calls GetStaticMethodID()which is actually Kaffe_GetStaticMethodID()to get the method handle for main()).
- build an array of strings as the arguments to the Java code's main()
  - * call the main method (CallStaticVoidMethod() on [10]),which will cause the entire program (or the executed portion thereof)to be translated into native code a function at a time.
- check for errors and then exit.

END

6. Conclusion

We provide an architecture that uses a Java class and native methods inside Kaffe environment for interfacing with embedded processors. We use the specific API plug-in on Kaffe libraries to write and read a value that contains the address of the board. These functions are implemented via the Java Native Interface (JNI) and Kaffe Native Interface (KNI) with Java-C integration. The goal of this integration is to design and port a Kaffe environment on an embedded processor with clear definitions of JNI and KNI.

This research demonstrates that a Kaffe co-processor is a practical and effective solution to Java's performance problems.

Acknowledgments. This work is partially supported by the Vietnam National IT Fundamental Research grant 204006.

References

[1] Didier Donsez, *Programmable Architecture on Java 2 Micro Edition*, 2006.
[2] Naveed Ahmad, Saddaf Mumtaz, *Software architecture of Kaffe*, 2006.
[3] J.Fleischmann, *A Hardware/Software Prototyping Environment for Dynamically Reconfigurable Embedded Systems*, 1998.
[4] Kenneth B.Kent, Micaela Serra, *Hardware Architecture for Java in a Hardware/Software Co-Design of the Virtual Machine*, 2002.
[5] The Java Chip Processor: Redefining the Processor Market, *Sun Microsystems*, November 1997.
[6] Stephan Schulz, Jerzy W.Rozenblit, Michael Mrva, Klaus Buchenrieder, *Model-based Codesign*, August 1998.
[7] Samuel K.Sanseri, *Porting Kaffe to a New Architecture*, 2000.
[8] Kiyo Inaba, *"What is trampoline code in Kaffe?"*, 1998. The URL at time of this writing is, http://www2.biglobe.ne.jp/ inaba/trampolines.html
[9] Azzam Mourad, *A Selective Dynamic Complier for Embedded JVM Targeting ARM Processors*, 2005.
[10] Kiyo Inaba, *"How sysdepCallMethod works"*, 1998. The URL at time of this writing is, http://www2.biglobe.ne.jp/ inaba/sysdepCallMethod.html