Asian Journal of Electrical Sciences (AJES)
Vol.2.No.1 2014 pp 11-15.
available at: www.goniv.com
Paper Received :08-03-2014
Paper Accepted:22-03-2013
Paper Reviewed by: 1. R. Venkatakrishnan 2. R. Marimuthu
Editor : Prof. P.Muthukumar

WCET ANALYSIS ON REAL TIME EMBEDDED SYSTEMS FOR MEMORY CONSTRAINS

Balachandar Jayapalan, Karthikeyan R,
Electronics and Communication Engineering
Arulmigu Meenakshi Amman College of Engg.
Thiruvannamalai, Tamil Nadu, India
balu26.2007@gmail.com, jaisanthini@gmail.com

ABSTRACT

The growing density of integration and the increasing percentage of system-on-chip memory occupied by embedded programs have led to an increase in the expected amount of power consumption. In order to reduce the integrity and iterations of the embedded programs the WCET has been implemented. In this paper, a compiler level optimization, namely WCET-aware rescheduling register allocation, is proposed to achieve WCET minimization for real-time embedded systems. The novelty of the proposed approach is that the effects of register allocation, instruction scheduling, and cluster assignment on the quality of generated code are taken into account for WCET minimization. These three compilation processes are integrated into a single phase to obtain a balanced result. By monitoring the Worst Case Execution time we can reduce the clock cycles required by each instruction of the program, which parallely increase the memory consumption based on both RAM and ROM memory in the embedded system and also power consumption criteria.

1. INTRODUCTION

Embedded systems play an important role in many areas of human life. Cell phones, PDAs, and satellites are only few examples of devices with a processor embedded in them. A large group of these systems are portable battery powered devices that have a limited source of energy. This makes the energy consumption a prominent characteristic. Hence, energy estimation during design phase of these applications helps designers in optimizing the energy consumption and the battery lifetime. Since software is responsible for a large portion of the system energy consumption, an accurate energy model is necessary for the system energy optimization.

Instruction-level Parallelism (ILP) is a critical technique used in computer architecture for processor and compiler design. ILP can improve the program execution performance by causing individual machine operations to execute in parallel. There are two modes of operation, sequential mode and parallelism mode. Macros and Pointers are two important functions which is used to estimate the memory size of RAM and ROM. Macros is used for such as addresses and array where the pointers are using without addresses that is addresses are hold in this process. In this process to avoid the worst case of execution (WCET).

To reduce WCET, a program is the maximal time execution can ever exhibit. Since WCET is one of the most important attributes of real-time systems, compiler level optimization of a program P should be conducted along the Worst-Case Execution Path. In addition, considering the characteristics of the clustered VLIW architecture, it is important to take into account the phase ordering problem of register allocation, scheduling and cluster assignment for WCET reduction so that a program WCET can be minimized.

2. EXISTING SYSTEM

Embedded systems frequently have to meet real-time constraints making them real-time systems. Without knowledge about the worst-case timing of a real-time application, designers tend to oversize hardware in order to guarantee that timing constraints are met. Knowing the worst-case execution time (WCET) thus enables to use or to design a hardware platform tailored towards the software resource requirements like memory or clock rate. Thus,

production costs can be reduced heavily while still guaranteeing the safeness of the real-time system.

Today, software development for embedded systems relies on high-level languages like C, and compilers. Modern compilers include a vast variety of optimizations. However, they mostly aim at minimizing e.g. average-case execution times (ACET). The effect of optimizations on WCET is almost fully unknown. Currently, the executable produced by the compiler is manually fed into a WCET analyzer computing timing information. Using this WCET data, it can be seen whether real-time constraints are met. If not, the code has to be tuned, compiled and optimized again in another cycle of the design flow.

Hence, it is desirable to have a WCET-aware compiler. An integrated WCET-aware compiler allows to integrate and to apply optimizations for WCET minimization. WCET data available within the compiler can be used to determine the worst-case execution path of a program. Specialized optimizations could be applied only to these code portions to minimize WCET aggressively.

3. PROPOSED SYSTEM

The execution time of a program is largely determined by its control flow. Here, control flow means the execution order of instructions or basic blocks, as represented by a program's control flow graph. In typical programs, control flow is expressed using constructs like e.g. loops or conditional branches. The worst-case execution time (WCET) of a computational task is the maximum length of time the task could take to execute on a specific hardware platform.

In general, static WCET analysis is undecidable since it is undecidable to compute how many times a general loop iterates. Since loop iteration counts are crucial for a precise WCET analysis, and since they cannot be computed for arbitrary loops in general, loop iteration counts need to be specified by the user of a WCET analyzer.

Besides loops known from high-level programming languages, any circle within a program's control flow graph needs to be annotated manually by the user. Any such user-provided annotations specifying the control flow are usually called flow facts.

The performance of today's systems is not constrained by the computing power of processors. Instead, the memory subsystem forms a bottleneck slowing down fast, modern processors. Particularly in the domain of hard real-time systems, such a slowdown is unacceptable. As a consequence, various optimizations within WCC aim at efficiently exploiting memory hierarchies by moving portions of a program's code and data to fast memories

Worst case execution time is typically used in reliable real-time systems, where understanding the worst case timing behaviour of software is important for reliability or correct functional behaviour.

As an example, a computer system that controls the behaviour of an engine in a vehicle might need to respond to inputs within a specific amount of time. One component that makes up the response time is the time spent executing the software – hence if the software worst case execution time can be determined, then the designer of the system can use this with other techniques such as schedulability analysis to ensure that the system responds fast enough.

While WCET is potentially applicable to many real-time systems, in practice the an assurance of WCET is mainly used by real-time systems that are related to high reliability or safety. For example in airborne software, some attention to software is required by DO178B. The increasing use of software in automotive systems is also driving the need to use WCET analysis of software.

Both of these techniques have limitations. End to end measurements place a high burden on software testing to achieve the longest path counting instructions is only applicable to simple software and hardware. In both cases a margin for error is often used to account for untested code, hardware performance approximations or mistakes. A margin of 20% is often used, although there is very little justification used for this figure, save for historical confidence.

As software and hardware have increased in complexity, it has driven the need for tool support. Complexity is increasingly becoming an issue in both static analysis and measurements. It is difficult to judge how wide the error margin should be and how well tested the software system is. System safety arguments based on a high-water mark achieved during testing are widely used, but become harder to justify as the software and hardware are less predictable.

4. IMPLEMENTATION

Memory Hierarchy

To enable optimizations moving parts of a program across memories, the information which is usually available only to the linker in a conventional compilation process needs to be provided already to the WCC compiler itself. This is motivated by the fact that

the WCET analyzer integrated into WCC requires detailed information about a program's memory layout. The stand-alone implementation used a fully linked and relocated binary executable providing the entire memory layout with its binary image. Now that is used as an integral part of the WCC compilation framework, linking and memory layout of programs need to be considered by the compiler.

Memory Allocation

Many architectures are equipped with fully software-controllable secondary memories. These are memories that are tightly integrated with the CPU to achieve best possible performance. These scratchpad memories (SPMs) can be accessed directly and are therefore in general well-suited for optimizations regarding energy consumption and execution times.

For WCET-centric optimizations, a scratchpad memory is ideal since the timing of such memories is fully predictable. Within the WCC compiler, SPMs are exploited for WCET minimization by placing assorted parts of a program into a scratchpad memory

Memory Architecture Compilation

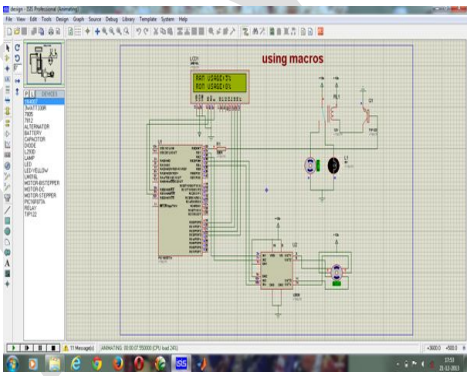
During the recent years, the speed of processing has been increasing significantly faster than the speed of memories. Therefore, many high-performance applications are typically constrained by the speed of the memory system. This effect has been called the memory wall the path toward higher performances is blocked by the limited speed of memories. Problems resulting from this fact can partially be reduced by exploiting the principle of locality and memory hierarchies. According to the principle of locality, real applications do typically exhibit some locality in the way in which they access memory. Memory hierarchies contain small, fast memories as well as larger, slower memories. Typically, caches are used in such hierarchies. For embedded systems, caches come with some disadvantages: their timing behaviour is difficult to predict and may exhibit a large variance. Also, caches are power-hungry. We proposed using scratchpad memories. Such memories behave like a kind of software-managed cache, replacing or complementing the usual hardware-managed cache.

In a more general context, we try to find transformations of embedded software exploiting the memory architecture in general. Toward this end, we make the memory architecture visible to optimization tools. These tools are then expected to modify the software such that improvements in terms of average or worst case execution times, energy consumption or memory footprints are achieved. The vision is to enable any software designer to optimize his/her software for the memory architecture at hand

5. RESULTS ANALYSIS

Macros Programming

A macro in computer science is a rule or pattern that specifies how a certain input sequence should be mapped to a replacement input sequence according to a defined procedure. The mapping process that instantiates a macro use into a specific sequence is known as macro expansion. A facility for writing macros may be provided as part of a software application or as a part of a programming language. In the former case, macros are used to make tasks using the application less repetitive. In the latter case, they are a tool that allows a programmer to enable code reuse or even to design domain-specific languages.



Macros are used to make a sequence of computing instructions available to the programmer as a single program statement, making the programming task less tedious and less error-prone. Macros often allow positional or keyword parameters that dictate what the conditional assembler program generates and have been used to create entire programs or program suites according to such variables as operating system, platform or other factors.

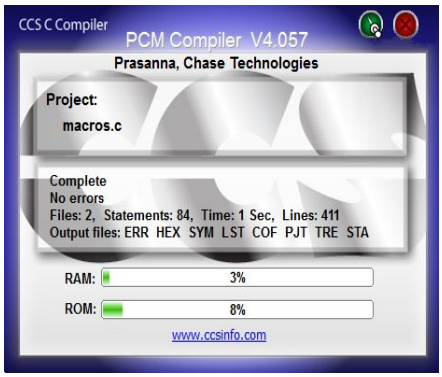
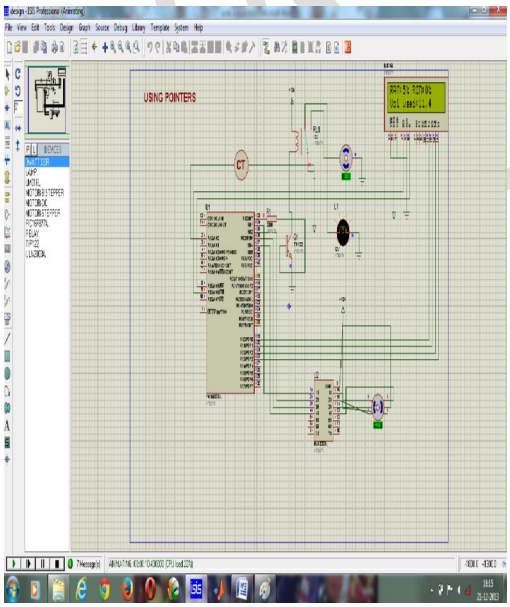


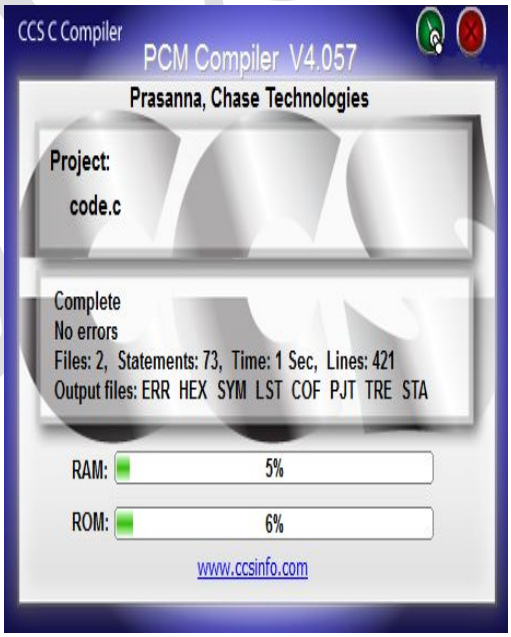
Figure . 1Pointers Programming

A pointer is a programming language data type whose value refers directly to another value stored elsewhere in the computer memory using its address. For high-level programming languages, pointers effectively take the place of general purpose registers in low-level languages such as assembly language or machine code, but may be in available memory. A pointer references a location in memory, and obtaining the value stored at that location is known as dereferencing the pointer. A pointer is a simple, more concrete implementation of the more abstract reference data type. Several languages support some type of pointer, although some have more restrictions on their use than others. As an analogy, a page number in a book's index could be considered a pointer to the corresponding page dereferencing such a pointer would be done by flipping to the page with the given page number.

Pointers to data significantly improve performance for repetitive operations such as traversing strings, lookup tables, control tables and tree structures. In particular, it is often much cheaper in time and space to copy and dereferences pointers than it is to copy and access the data to which the pointers point. Pointers are also used to hold the addresses of entry points for called subroutines in procedural programming and for run-time linking to dynamic link libraries (DLLs). In object-oriented programming, pointers to functions are used for binding methods, often using what are called virtual method tables.



While pointer has been used to refer to references in general, it more properly applies to data structures whose interface explicitly allows the pointer to be manipulated as a memory address, as opposed to a magic cookie or capability where this is not possible. Because pointers allow both protected and unprotected access to memory addresses, there are risks associated with using them particularly in the latter case. Primitive pointers are often stored in a format similar to an integer however, attempting to dereference or look up a pointer whose value was never a valid memory address would cause a program to crash. To alleviate this potential problem as a matter of type safety, pointers are considered a separate type parameterized by the type of data they point to even if the underlying representation is an integer. Other measures may also be taken. It can be faster and can incur less overhead, both in data structures and in keeping the program execution footprint down. It gives you much more raw access, and this can be very helpful, clever, or necessary. You can point to anywhere and treat it pretty much as anything. Generally useful in the context where we need a continuous memory allocation. Using pointers dynamic allocation of memory is achieved. Pointers basically hold the address of a variable. they are mainly used as function parameters to pass values of parameters as references rather than values.



6. CONCLUSION

This paper proposes a compiler level optimization technique, namely WCET-aware re-scheduling register allocation for WCET reduction on real-time embedded systems with instruction level parallelism. A processor that executes every instruction one after the other i.e. a non-pipelined scalar architecture may use processor resources inefficiently,

potentially leading to poor performance. The performance can be improved by executing different sub-steps of sequential instructions simultaneously this is pipelining or even executing multiple instructions entirely simultaneously as in superscalar architectures. Further improvement can be achieved by executing instructions in an order different from the order they appear in the program.

FUTURE ENHANCEMENT

By using macros and pointers programming we achieve WCET minimization in compiler level optimizations. The future work of this project is to implementation in hardware by using scheduling and compare normal programming and RTOS programming to obtain the best result.

REFERENCES

[1]. Yazhi Huag, Liang Shi, Jianhua Li, Qingan Li, and Chun Jason Xue "WCET-Aware Re-Scheduling Register Allocation for Real-Time Embedded Systems With Clustered VLIW Architecture", IEEE transactions on Very Large Scale Integration Systems.

[2]. D. Sciuto, C. Silvano and V.Zaccaria(2012), "An Instruction-Level Energy Model for Embedded VLIW Architectures", IEEE Transcations.

[3]. G. Callou, P. Maciel, E. Tavares, E. Andrade, B. Nogueira, C. Araujo, and P. Cunha(2011), "Energy Consumption Estimation in Embedded Systems"

[4]. H. Falk. "WCET-aware register allocation based on graph coloring,"in *DAC '09: Proceedings of the 46th annual design automation conference,*2009, pp. 726–731.

[5]. "MAP1000 unfolds at equator," in *Microprocessor Report*, 1998.J. Fridman and A. Greefield, "The TigerSharc DSP architecture," in*IEEE Micro*, 2000, pp. 66–76.

[6]. M. D. Smith, N. Ramsey, and G. Holloway, "A generalized algorithm for graph-coloring register allocation," in *PLDI '04: Proceedings of the ACM SIGPLAN 2004 conference on programming language design and implementation*, 2004, pp. 277–288.