INTERNATIONAL JOURNAL OF HUMAN COMPUTING STUDIES

https://journals.researchparks.org/index.php/IJHCS e-ISSN: 2615-8159 | p-ISSN: 2615-1898

Volume: 04 Issue: 7 | Jul 2022

Advances in Compiler Construction for Adaptive Computers

Benjamin, Ubokobong Effiong

Department of Computer Science, Akwa Ibom State Polytechnic, Ikot Osurua, Ikot Ekpene

benjaminubokobong52@gmail.com

----------***----------

Abstract: This paper is focused on advances in compiler construction for adaptive computers. Compiler construction is the process of constructing compilers and a compiler converts a source code into a machine code for implementation. A classical compiler is targeted to a specific hardware and processor configuration while an adaptive compiler adapts to different hardware and processor configurations making it very efficient. The term compilation denotes the conversion expressed in a human oriented source language to an equivalent algorithm expressed in a hardware oriented target language. Adaptive systems, on the other hand, accelerate programs by executing parts of the algorithm on adaptive hardware. These elements can be dynamically reconfigured during the program run. This has given rise to adaptive computers. Adaptive compiler is therefore responsible for possibility of adaptive computer systems. This makes research on adaptive compiler construction for adaptive computers important. The aim of this paper is therefore to reveal how adaptive compiler construction has influenced the development of adaptive computers.

Keywords: Adaptive, compiler, re-configuration, adaption, programming.

Introduction

A compiler is a program which converts high level programming language into low level programming language or converts source code into machine code. Compilers and operating systems constitute the basic interfaces between a programmer and the machine making. The process of building compilers is called compiler construction. Compilers are usually built to run on specific machine architecture. To overcome this problem, there is need to build compilers that adapt to different machines, thereby paving way for a new generation of compilers, self-re-configurable compilers. This implies that this kind of compiler adapts to different machine architecture (Mahak, 2014).

The term "adaption" in computer science refers to a process, in which an interactive system or adaptive system adapts its behaviour to individual users based on information acquired about its user(s) and its environment. Adaptive systems accelerate programs by executing parts of the algorithm on adaptive hardware. Processors have grown more complex, with multiple functional units, exposed pipelines and myriad latencies that must be managed. In most cases, these computers execute code produced by compilers – a translator that consumes source code and produces equivalent for some target machine.

These elements can be dynamically reconfigured during the program run. Some research projects in adaptive systems have already demonstrated the advantages. At the same time, the application of computing to new problems has created demand for compilers that optimize programs for new criteria, or new objective functions. For most modern processors, optimizing compilers can be built that produce efficient code for a single uniprocessor target. This development of developing adaptive compilers has birthed adaptive computing.

INTERNATIONAL JOURNAL OF HUMAN COMPUTING STUDIES

https://journals.researchparks.org/index.php/IJHCS e-ISSN: 2615-8159 | p-ISSN: 2615-1898

Volume: 04 Issue: 7 | Jul 2022

Therefore, adaptive compilers ease the inevitable transitions to new hardware and programming languages. The software part is instrumented with functions for configuring and exchanging with the reconfigurable hardware. Adaptive computing refers to the capability of a computing system to adapt one or more of its properties (e.g. performance) during runtime. There are diverse reasons of why it is advantageous for a computing system to adapt during runtime and there are various enabling techniques and paradigms that allow a computing system to perform such an adaptation. In the following, we limit our explanations of adaptive computing systems to the newest advances in embedded computing systems. Often, reconfigurable computing is referred to as adaptive computing. Actually, it corresponds to one of the key paradigms that along with application-specific instruction set processors enable adaptive computing. We will shortly discuss state-of-the-art in both areas (Oppermann, 2005).

Also known as "reconfigurable computing," it refers to a logic chip that can change its physical circuitry on the fly. Evolved from programmable architectures such as complex programmable logic device (CPLD) and field-programmable gate arrays (FPGA), adaptive computing is an order of magnitude faster in rate of reuse (ROR) and can reconfigure itself in nanoseconds. Fast Hardware Reconfiguration Primarily designed for the cell phone and wireless market, adaptive chips use very little power and can process different types of algorithms in the same circuit space. For example, rather than requiring a dedicated circuit for error correction and another for decompression, the adaptive chip dynamically reconfigures itself for each algorithm as needed. In addition, when new algorithms are invented, instead of designing a new Application Specific Integrated Circuit (ASIC), the adaptive chip is given new instructions to load. This development forms the basis of operation of adaptive computers (Henkel and Parameswaran, 2007).

Concept of Compiler Construction

The core compiler reads a program described in a high-level programming language. The compiler then analyses the program, partitions it into hardware and software, and then generates data paths for the reconfigurable hardware. It focuses on the basic relationships between languages and machines. The term compilation denotes the conversion of an algorithm expressed in a human-oriented source language to an algorithm expressed in a hardware-oriented target language. Also consider Conventional programs give priority to knowledge in which competency is flexible and adaptable and cannot be reduced to an algorithm. Programming languages are the tools used to construct formal descriptions consists of finite computations (algorithms), in which each computation further consists of operations that transform a given initial state into the final state. In the context of factual information that can consist of, for example, a definition, a theorem, a hypothesis, a rule, or an algorithm. In engineering compilers, the following is needed (Jalgeri, Jagadale and Navale, 2017).

Lexical Analysis: The lexical syntax (token structure), which is processed by the lexer and the phrase syntax, which is processed by the parser. The lexical syntax is usually a regular language, whose alphabet consists of the individual characters of the source code text. The phrase syntax is usually a context-free language, whose alphabet consists of the tokens produced by the lexer. In computing, lexical analysis is the process of converting a sequence of characters into a sequence of tokens, i.e. meaningful character strings. A program or function that performs lexical analysis is called a lexical analyzer, lexer, tokenizer, or scanner, though "scanner" is also used for the first stage of a lexer.

Parser: Within computational linguistics the term is used to refer to the formal analysis by a computer of a sentence or other string of words into its constituents, resulting in a parse tree showing their syntactic relation to each other, which may also contain semantic and other information. The term has slightly different meanings in different branches of linguistics and computer science. In order to parse natural language data, researchers must first agree on the grammar to be used. The choice of syntax is affected by both linguistic and

https://journals.researchparks.org/index.php/IJHCS e-ISSN: 2615-8159 | p-ISSN: 2615-1898
Volume: 04 Issue: 7 | Jul 2022

computational concerns; traditional sentence parsing is often performed as a method of understanding the exact meaning of a sentence, sometimes with the aid of devices such as sentence diagrams. It usually emphasizes the importance of grammatical divisions such as subject and predicate. Parsing or syntactic analysis is the process of analysing a string of symbols, either in natural language or in computer languages (Potluri, Konga and Kadiyala, 2014).

Lexer-Parser Processing: While using a lexical scanner and a parser together, the parser is the higher level routine. These generators are a form of domain-specific language, taking in a lexical specification – generally regular expressions with some mark-up and outputting a lexer. The lexer then scans through the input recognizing tokens. Automatically generated lexer may lack flexibility, and thus may require some manual modification or a completely manually written lexer. The next stage is parsing or syntactic analysis, which is checking that the tokens form an allowable expression. This is usually done with reference to a context-free grammar which recursively defines components that can make up an expression and the order in which they must appear. However, not all rules defining programming languages can be expressed by context-free grammars alone, for example type validity and proper declaration of identifiers. These rules can be formally expressed with attribute grammars. This can be done in essentially two ways:

A. ***Top-down parsing***- Top-down parsing can be viewed as an attempt to find left-most derivations of an input-stream by searching for parse trees using a top-down expansion of the given formal grammar rules.

B. ***Bottom-up parsing*** - A parser can start with the input and attempt to rewrite it to the start symbol. Intuitively, the parser attempts to locate the most basic elements, then the elements containing these, and so on. LR parsers are examples of bottom-up parsers. Another term used for this type of parser is Shift-Reduce parsing.

Recent Advances in Compiler Construction

The newest advances in compiler construction are based on the development of adaptive compilers for adaptive computing. Adaptive computing build upon the paradigms in state-of-the-art reconfigurable computing and the design of application-specific instruction set processors (ASIP). The properties of many of today's embedded systems are often hard to predict at design time because of their complexity (the extent of supported functionality) in conjunction with (at design time) hard-to-predict user profiles. Typical examples are consumer devices like cell phones that offer a large variety of functionality, e.g. web access, navigation, gaming, video/audio streaming etc. Design constraints include high performance at low cost, low power consumption etc. A further field of application for adaptive computing is to increase dependability (Neumann, Sydow, Blume and Noll, 2008).

How can an embedded system adapt its computing properties during runtime? This can only be achieved with adaptive compilers. The compiler/application developer is responsible for determining the configuration that shall be loaded at a particular time. The WARP processor offers a high potential for adaptivity by determining and synthesizing suitable hardware accelerators during runtime, which allows adapting to various scenarios (Vahid, Stitt and Lycesky, 2008). Adaptive Computing will gain further popularity as systems will become more complex and less predictable (Neumann, Sydow, Blume and Noll, 2008).

Design decisions that were formerly carried out during design time and compile time now need (at least partially) to be shifted to runtime. The runtime system, as part of the processor, decides on the fly which sub-set of the pre-defined special instruction set is invoked during runtime and in which flavor it is executed (i.e. implementation version with certain computational properties). Partial runtime reconfiguration is used to adapt the special instruction set hosted in the embedded FPGA. A runtime monitoring system evaluates adaptations and it may improve adaptations in subsequent execution iterations. For instance, the Molen

https://journals.researchparks.org/index.php/IJHCS e-ISSN: 2615-8159 | p-ISSN: 2615-1898
Volume: 04 Issue: 7 | Jul 2022

processor offers a set of control instructions that are used to determine the reconfigurations and to execute special instructions (Vasiliadis, 2004).

Concept of Adaptive Computers and Adaptive Computing

Adaptive Computers are computers that implement adaptive computing technology. Adaptive computing refers to the capability of a computing system to adapt one or more of its properties (e.g. performance) during runtime. There are diverse reasons of why it is advantageous for a computing system to adapt during runtime and there are various enabling techniques and paradigms that allow a computing system to perform such an adaptation (McCluskey, 2000). Often, reconfigurable computing is referred to as adaptive computing. Reconfigurable computing deploys a reconfigurable fabric to implement user-defined functionality. Modern field-programmable gate arrays (FPGAs) provide a capacity of millions of ASIC gate equivalents and are composed of clustered lookup tables and switching matrices FPGAs provide flexible fine-grained reconfiguration. They can be used to prototype architectures at an early development stage. FPGAs furthermore allow so-called partial runtime reconfiguration, which means to partly reconfigure during operation (requiring approx. 1 to 10 ms) while other parts remain unaffected. Realizing adaptive computing by means of embedded FPGAs is possible (Hauck and DeHon, 2008).

Another paradigm enabling adaptive computing hence adaptive computers are Application Specific Instruction Set Processors (ASIPs). Whereas the term ASIPs in the early 1990s denoted processors with a domain-specific (e.g. multi-media) instruction set that is designed after a careful analysis of a certain domain or even a specific application, in the late 1990s ASIP tool suites like evolved that allow the designer to tailor highly specific processors. Customization in typical ASIP tool suites may be accomplished through one or more of the following options:

i) Instruction set extension: a customized instruction set that is defined by means of an architectural description language (ADL), then synthesized and enhancing the fixed (i.e. pre-defined) core instruction set of an embedded processor;

ii) Inclusion/exclusion of predefined blocks: factory pre-designed custom blocks like special register sets, caches etc. may be included as an option for further customization;

iii) Parameterization: Various parameters like size of caches, bit-width etc. allow for further tuning

In addition, some ASIP tool suites allow designing the entire instruction set architecture from scratch using an ADL. An enabling technique for this kind of flexibility in designing embedded processors is retargetability: it refers to the capability of a development tool, for instance a compiler, to generate optimized code for a customized ASIP (Schliebusch, Meyr and Leupers, 2007).

Compiler Construction Influence on Adaptive Computers

One of the key challenges of future multi-core adaptive computers architectures is programmability and scalability. As compilers link the code written by programmers to the underlying parallel hardware, this cluster has a pivotal role to play. It will focus on versatility by adapting the user code to the ever changing underlying hardware. It will pursue a system wide perspective on adapting programs, and more generally workloads, to both short-term architecture variation, such as cache miss, and longer-term changes, such as the increasing number of processors available (Thoma, 2007).

The choice of specific transformations and an order for their application play a major role in determining the effectiveness of an optimizing compiler. An ordered list of transformations is called a compilation sequence. Since the 1960s, complier writers have chosen compilation sequences in an ad hoc fashion, guided by

https://journals.researchparks.org/index.php/IJHCS e-ISSN: 2615-8159 | p-ISSN: 2615-1898
Volume: 04 Issue: 7 | Jul 2022

experience and limited benchmarking. Efforts to find the best sequence have foundered due to the complexity of the problem. Transformations both create and suppress opportunities for other transformations. Different techniques for the same problem catch different subsets of the available opportunities. Finally, combinations of techniques can achieve the same result as some single techniques (Cooper, Subramanian and Torczon, 2002).

Instead of translating directly into machine code, modern compilers translate to a machine independent intermediate code in order to enhance portability of the compiler and minimize design efforts. The intermediate language defines a virtual machine that can execute all programs written in the intermediate language. The intermediate code instructions are translated into equivalent machine code sequences by a code generator to create executable code. It is also possible to skip the generation of machine code by actually implementing the virtual machine in machine code. This virtual machine implementation is called an interpreter, because it reads in the intermediate code instructions one by one and after each read executes the equivalent machine code sequences of the read intermediate instruction directly (Vahid, Stitt and Lysecky, 2008).

The use of intermediate code enhances portability of the compiler, because only the machine dependent code of the compiler itself needs to be ported to the target machine. The remainder of the compiler can be imported as intermediate code and then further processed by the ported code generator, thus producing the compiler software or directly executing the intermediate code on the code generator. The machine independent part can be developed and tested on another machine. This greatly reduces design efforts, because the, machine independent part needs to developed only once to create portable intermediate code.

Adaptive computers refer to special versions of already existing computers or tools that provide enhancements or different ways of interacting with the technology.

Examples of adaptive computers applications are:

- ✓ Large print books
- ✓ Digitized text
- ✓ Good lighting
- ✓ Large monitors
- ✓ Software to adjust screen colors
- ✓ Computers with voice output
- ✓ Computers with visual output
- ✓ Electronic mail
- ✓ Word prediction software
- ✓ Adjustable tables
- ✓ Keyboard modifications

The adaptation helps individuals with a disability or impairment accomplish a specific task. Adaptive technology also includes what is known as “assistive technology.” This term refers to any light-, mid-, or high-tech tool or device that helps people with disabilities perform tasks with greater ease and/or independence. Examples include:

https://journals.researchparks.org/index.php/IJHCS e-ISSN: 2615-8159 | p-ISSN: 2615-1898
Volume: 04 Issue: 7 | Jul 2022

- ✓ Screen readers
- ✓ Magnification applications
- ✓ Text-to-speech synthesizers
- ✓ Alternative keyboards
- ✓ On-screen keyboards
- ✓ Keyboard filters
- ✓ Electronic pointing devices
- ✓ Sip-and-puff systems
- ✓ Wands and sticks
- ✓ Joysticks
- ✓ Trackballs
- ✓ Touch screens
- ✓ Braille embossers
- ✓ Refreshable braille displays
- ✓ Light signaler alerts

Conclusion

Adaptive computers are the future of computing, more efforts is therefore needed in the development of adaptive compilers. Building a robust adaptive optimizing compiler require strategies to minimize explicit, user-selected objective functions over a complex and poorly characterized space—viz., the combinatorial set of distinct compilation sequences. It requires new techniques for implementing optimizations in a modular fashion, managing their reconfiguration, and measuring the results of each compilation. It will require careful consideration of stopping criteria, of strategies for producing and storing results incrementally, and of how to apply the knowledge gained by exploring the space of compilation sequences.

Recommendations

- ➢ More research on compiler construction should be encouraged.
- ➢ Adaptive compiler construction should be the new area of focus to promote development of adaptive computers.
- ➢ Experts on adaptive compiler construction should be contracted to train others.
- ➢ More adaptive computer problems should be identified for solution to be provided.

References

1. Cooper, K., Subramanian, D. Torczon, L. (2002). Adaptive Optimizing Compilers for the 21*st* Century. *Journal of Supercomputing*, pp. 1-14.
2. Hauck, S. and DeHon, A. (2008) Reconfigurable Computing: The Theory and Practice of FPGA-Based Computing, Morgan Kaufmann.

https://journals.researchparks.org/index.php/IJHCS e-ISSN: 2615-8159 | p-ISSN: 2615-1898
Volume: 04 Issue: 7 | Jul 2022

3. Henkel, J. (2003) .Closing the SoC Design Gap., *IEEE Computer Magazine*, Volume 36, Issue 9, pp. 119-121, Sept. 2003.
4. Henkel, J. And Parameswaran, S. (2007), .*Designing Embedded Processors* -- A Low Power Perspective., Springer,.
5. Jalgeri, A., Jagadale, B. and Navale. R. (2017). Study of Compiler Construction. *International Journal Of Innovative Trends In Engineering (IJITE)*, Issue: 46, Volume 28, pp. 86-88.
6. Mahak, J. (2014). Compiler basic design and construction. *International Journal of Computer Science and Mobile Computing*, Vol.3 Issue.10, October-, pg. 850-852
7. McCluskey, E. J. (2000) Dependable Computing and Online Testing in Adaptive and Configurable Systems. *IEEE Design & Test*, Vol. 17, Issue 1, pp. 29-41.
8. Neumann, B., von Sydow, T., Blume, H. and Noll, T. (2008) Design flow for embedded FPGAs based on a flexible architecture template, *Conference on Design, automation and test in Europe*, pp. 56-61,.
9. Oppermann, Reinhard (2005). User-Adaptive to Context-Adaptive Information Systems (PDF). I-com *Zeitschrift für interaktive und kooperative Medien*. **4** (3): 4–14.
10. Potluri, C., Konga, S. & Kadiyala, S. (2014). Technology in Construction of Compiler for Adaptive Computers. *International Journal of Scientific & Engineering Research*, Volume 5, Issue 11.
11. Schliebusch, O., Meyr, H. And Leupers, R. (2007). Optimized ASIP Synthesis from Architecture Description Language Models. Springer.
12. Thoma, F. (2007), MORPHEUS: Heterogeneous reconfigurable computing, *Intl Conf. on Field-Programmable Logic and Applications (FPL)*, pp. 409—414
13. Vahid, F., Stitt, G. and Lysecky, R. (2008) .Warp Processing: Dynamic Translation of Binaries to FPGA Circuits., *IEEE Computer*, Vol. 41, No. 7, pp. 40-46.
14. Vassiliadis, S. (2004) .The Molen Polymorphic Processor., *IEEE Transactions on Computers*, Vol. 53, Issue 1, pp. 1363-1375.