Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 78 (2016) 831 – 837

Procedia Computer Science

International Conference on Information Security & Privacy (ICISP2015), 11-12 December 2015, Nagpur, INDIA

Dependency Analysis for Secured Code Level Parallelization

Vandana Jagtap[a], , Urmila Shrawankar[b]

[a] *Research Scholar, SGB Amravati University, Amravati, India*
[b] *G. H.. Raisoni College of engineering, Nagpur, India*

Abstract

Today central topic in science and engineering is parallel and distributed computing, research performing in the area of development of new approaches for the modeling, design, analysis, evaluation, and programming of future parallel and distributed computing systems and its applications. To detect the presence of dependency between tasks, analytic methodology is required. This review theoretically analyzes what is dependency, different types of dependencies, various applications which require dependency analysis and different approaches towards dependency analysis. This research classifies dependency analysis approaches on the basis of input source; for different input source different methodology is available, this review considers an input source as a component base system with dependent components. This research gives review of dependency analysis solutions for various areas like parallel processing, high performance computing, security and vulnerability in software's where various approaches are used for dependency analysis. This review concludes that there are various methods available for dependency analysis. Most of the methods use graph base approach, conceptual graph approach and matrix based approach, out of which this review puts forward the best possible approach. This research proposes a new approach for dependency analysis which results in the reduced dependent analysis time, low memory consumption and less cost sensitive.

Peer-review under responsibility of organizing committee of the ICISP2015

Keywords: Data dependency; dependency type; parallel Computing; high performance computing; dynamic analysis; directed acyclic graph.

1. Introduction

As the social universality of automated systems increases, the demand for new systems and enhancements to existing systems also increases. So the application level analysis of system is required. Dependency analysis is concerned about dependencies due to interconnections between system components. Dependency analysis is also an important aspect of any parallel programming tool. Its area has served as grounds for fruitful research in parallel

Peer-review under responsibility of organizing committee of the ICISP2015
doi:10.1016/j.procs.2016.02.067

computing, high performance computing and software engineering.

In parallel computing two issues are essentially concerned and equally important is the partitioning method as well as scheduling and distribution policy of subtasks (reordering). Security is a major concern in scheduling and distribution. Dependency analysis is also used to compute user dependencies for security and vulnerability.

Definitions of dependency given in the literature are varying widely. One of the first definitions of dependency in the literature was proposed 40 years back by Stevens.et.al (1974) [1] as - a dependency is the degree to which each component relies on each one of the other components. Some sources mention that dependencies are simply first-order logic formulae. Some take a probabilistic approach for defining the dependencies and express dependencies as conditional probabilities between some specific variables. Some sources state an approach that a dependency is best modelled by using the client/server relationship, and then propose the definition of dependency in client/server terms. In general case, dependency is defined as a relation D [2], between some numbers of entities where in a change to one of the entity implies a potential change to the others as shown in Fig 1.

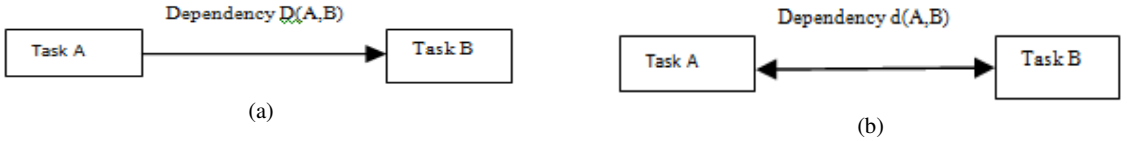

Fig1. Graphical representation of the dependencies
1(a) Simple dependency between two tasks; 1(b) Bi-directional dependency between two tasks

In case of parallel processing, more dependencies imply more access time or more inter-processor communication. This degrades the overall performance of the system. In software engineering, analysis of dependencies is very important in its every phase; from planning to design and also in maintenance of the system[3].

The structure of the paper is as follows: Section 2 gives an overview of classifications of types of dependencies. In section 3 gives application areas where dependency analysis is required. In section 4 discusses classification depending on the input source; for different dependency analysis approaches as code based, model based and run-time analysis. Section 5 gives an overview of existing available dependence analysis approaches and comparative study; in section 6 proposal of dependency analysis system. Section 7 discusses expected outcome and conclusion.

1.1. Motivating example

Example 1:

Statement 0: X=X1+X2
Statement 1: Y= Y1+Y2
Statement 2: Z= Z1+Z2

(a)

Example 2:

Statement 0: X=X1+X2
Statement 1: Y= Y1+Y2+X
Statement 2: Z= Z1+Z2
Statement 3: Q = X+Y+Z

(b)

Example illustrated in Fig. 2(a) demonstrates a code where every statement is depends on its predecessor; all statements have to be executed sequentially as there is no parallelism possible. In contrast code from Fig. 2(b) has ample amount of parallelism possible. Here two operations do not depend on any other operations and can be executed in parallel. In order to achieve parallelism, statements have to be reordered and scheduled. While reordering, correctness has to be maintained regarding dependencies. Here we used dependency analysis approaches.

Fig 2. Motivating example

2. Classifying dependencies

Different dependency types have been proposed in various dependency models over the years. They have different level of abstraction and criteria for categorization. For varying areas different dependency categorizations are available. Some sources specify types of dependency as structural and functional dependencies or data and value dependencies; similarly somewhere dependency types are classified into intrinsic and additional dependencies on the top level. This paper considers generic type of dependencies; which are classified into three broad categories that is structural dependency, behavioural dependency and traceability dependency.

2.1. *Structural dependency*

Structural dependencies allow one to locate source specifications that contribute to the description of some state or interaction. Several subcategories that come under structural dependency are Content Dependencies, Common

Dependencies, External dependencies, Control dependency and data dependencies.

2.2. Behavioural dependency

Behavioural dependencies consent one to relate states or interactions to other states or interactions. Behavioural dependency consists of abstractions which are not directly provided by programming languages; Example, external programs or devices, event broadcast, client-server protocols etc. Behavioural dependence analysis is traditionally used to perform risk analysis, fault tolerance and redundancy provisions in distributed real time systems. Traditionally, most behavioural dependencies are found by inspecting the source code directly or by analyzing execution traces of system [4].

2.3. *Traceability dependency*

Traceability supports the alignment between various stakeholder concerns, development artefacts and different products of the software development process. Developer cannot ignore requirements after the design is built nor can ignore the design after the source code is programmed, as software development is a continuous process[5]. Therefore it is required for developers to maintain inter-relationships between different artefacts. These inter-relationships are specifically called traceability dependencies.

3. Application areas where dependency analysis required

Dependency analysis addresses issues in various contexts of developing systems like in; Software engineering for program understanding, testing, debugging, reverse engineering[6] and maintenance. In natural language processing dependency analysis is required for parsing, construction of dependency tree and relation extraction (RE)[7]. In image processing for decoding image and video bit streams with maximum concurrency[8].In databases and data structures for data normalization[9]. In distributed services to identify how the failure of one process can potentially affect other concurrently executing processes. In signal processing for repetitive structure modelling (RSM) to inherit parallelism dependency analysis is required[10]. Dependency analysis is mainly required in compilers for code level parallelization, optimizing and parallelizing compilers.

4. Classification of dependency analysis approaches

Here classification of dependency analysis approaches is done according to their type of input source of information, which is different scale of granularity e.g. instructions, basic blocks, function calls, etc. Dependency analysis approaches take the source of information as input data and transform this information into high level abstract information. This high level information is then used to reason about the dependencies and to solve dependency issues in the various application areas. The information can be in the form of graph, matrix or table that can represent dependencies in a system. Depending upon input source existing analysis approaches are classified into three groups; code base approach, component base approach, and run-time analysis approach.

In first code based approach, analysis approach is applied to program or code (instructions); where relationships among statements, loops and variables in a program are considered. Source code based solutions can be applied to parallelize legacy sequential code for better utilization of processor, for parallelizing compilers and for natural language processing systems [11].

In second model base approach communication/interaction between models are considered. System is represented using model based representation that provides more specific information about the dependency. Model based approaches are used in parallelizing the systems, in automatically partitioning softwares [5], and in application level analysis and management of the system.

In third; run-time analysis approach, applications those require runtime monitoring of the objects to detect dependencies are considered. Both code base and model base approaches are used for runtime analysis [12].

4.1. Code base approaches

Code base is the most well known source used by dependency analysis solutions. Both syntactic and semantic information gives detail about the code components (e.g. variables, operations, methods, classes) and relationships among them. Source code based approaches are often used to analyze the structural dependencies at

different levels of abstractions like statement level and module level [13]. The various methods available are static analysis, dynamic analysis and historical analysis.

Table 1- Code based dependency analysis methods

Tests for code base analysis	Advantages	Limitations
GCD Test	Simple and efficient at disproving dependencies.	Inexact test, ignores loop limit and inequality constraints Does not provide direction vector information.
Banerjee Test	Simple test and efficient at disproving dependencies and generates direction vector information.	Approximate test It takes only single subscript of a multi-dimensional array.
I-test	Includes all the advantages of GCD and Banerjee test and extends the range of applicability of Banerjee test.	Constraint of each variable has to be integer, Cannot handle multi-dimensional array references involving coupled subscripts
PVI test	Applicable for nonlinear subscripts	Lowers efficiency when mixed polynomial exists
Quadratic programming test	Exact test and works efficiently for mixed polynomials.	Time complexity is high Coefficient matrix of quadratic terms should be positive semi-definite.

4.2. *Component based approach*

In Component base approach, mainly workflow of an application considered. Workflow of an application means the atomic computation units and their data dependencies. This approach takes models of system as an input; this model can be in the form of UML models or architectural description language (ADL), Interface description language (IDL) that describe structure and behaviour of system. Different kinds of graph methods are used for representing these kinds of systems, where the communication among functions in a program forms a graph, where nodes are represented by functions and edges shows communication among functions.

Table 2- Component based dependency analysis methods

Method used	Type of analysis	Highlights
Directed Acyclic Graph (DAG) and Hierarchical DAG	Static/ Dynamic	Used to represent dependencies between tasks and entities. Provides directions of the dependency relationship. Nodes represent task/entity and arcs represent dependency relation between nodes. Compared to normal DAG hierarchical DAG provides more accuracy.
Matrix Representation and Adjacent Matrix Representation	Static	Traditional graph theory combined with matrix representation is able to determine whether the components are dependent or not. Adjacent matrix is used to represent directed graphs.
Conceptual Graphs	Static, Dynamic	Provides more detail information about the dependence relation. It is scalable.
Data flow Graphs (DFG)	Static	It is usually very large and contains lot of inter dependencies.
Dynamic data flow Graphs (DDFG)	Dynamic	Graph constructed at runtime, is analyzed subset of the full graph with possible execution path and improves along with iterations of execution. It can exploit more parallelism.

4.3. Run-time analysis approach

In this approach execution information of code or component based system is considered for analysis. So there is an additional overhead required in gathering execution related information about the system. These approaches can identify dependencies without accessing the actual source code [14]. Run-time monitored information includes tracking of events that occur at runtime. This information can be generated by using some system infrastructure or with trace generator facilities. Runtime dependencies between features are difficult to spot by just inspecting the source code, as modifications performed on one part of code may affect other features.

5. Related work and comparative study

As discussed in the previous section there are many types of dependencies in various areas. There are various approaches like DAG, Matrix representations, databases, which are used to show communications and transmissions between models. Literature shows various varieties with graph approach and matrix approach some are mentioned below:

EasyPar[15] is an intelligent analysis engine for parallelization having unique features to assist the programmer at the time of program development. It uses query based data dependency analysis method for dependency analysis. This database approach has three benefits it makes incremental parsing easier and faster, efficient dependency analysis and demand driven program analysis. This method is not generalized for all algorithms. Parwiz[16] is a framework to capture all data dependencies for parallelization, where vector technique and call graph are used for dependence analysis. It analyzes dependency dynamically at run time. This combination is important for lowering the cost of empirical analysis but it is time and memory consuming methodology compared to query based data dependency analysis.

StarSs[17] is another programming model for parallel programming; it is able to detect task dependencies automatically. Another specific feature of the current CellSs/ SMPSs implementations is the automatic detection of data dependencies between tasks and runtimes classifying the type of data dependency like read after write (RaW), write after write (WaW) and write after read (WaR). It has overhead of additional usage of memory and processing time. This is not a suitable approach for an application consisting more of parallelization. Fully automatic tools have many limitations compared to semiautomatic or manual approach that is low efficiency and lack of interaction with the user.

Dependency can also be analyzed using metadata and its buffer dynamically at runtime. In [18] Potential thread-level-parallelism exploration with superblock reordering; dynamic data dependencies are analyzed. In this approach the dynamic data flow graph (DDFG) of sequential programs are analyzed for dependency analysis. DDFG exploits more parallelism. Some parallelism can only be identified by dynamic analysis.

Directed acyclic graph (DAG) method for finding dependencies between tasks is used in literature [19, 20]. DAG can be improved to the multilevel DAG, to hierarchically represent dependency relationships of data at different levels. Hierarchical DAG improves the performance of the system compared to simple DAG but additional spaces saving techniques are required to be integrated for improving performance of the system. DAG is also found to be good solution for scheduling of the task for parallel processing, load balancing, and spatial decomposition. On the similar lines for concurrency; literature [21] introduced special logic for true concurrency, which allows prediction of mutual dependencies between events.

Conceptual Graphs provide a powerful approach to represent, characterize, and analyze dependencies between the entities in a model base dependency analysis [22]. Dependencies are mapped into conceptual graphs. Fig 1 depicts a dependency in graphical form. Conceptual dependency graphs are relationally expanded to include attributes of dependency. A conceptual graph allows representation of more specific information about the dependency. One of the advantages of conceptual graphs representation is its scalability. It is useful for applications where we require more knowledge about dependency. There were many dependency types mentioned in the literature from various points of interest, but there is still lack of assessment in the applicability of different dependency types in software engineering [23].

Dependency is a represented using adjacency matrix in graph theory. This representation can check presence of dependency between the components. It does not consider type of interaction between components. One more dependency representation methodology is link-list based dependency representation [24]. This can be implemented using Hash Map in Java; which provides more information about the dependency relation. This information can be used to solve dependency related issues. This approach is more efficient and faster than matrix based approach.

6. Proposed system

The main module of proposed system includes trace collection, dependency analysis, and graph representation of analyzed code. Input code is parsed and split into tokens. These tokens contain information about program constructs. This information is then used for further analysis. The next phase obtains execution traces from the input code. The trace collector takes parsed program constructs and compiled code as an input and generates execution traces. These traces gives information about the run time occurred traces of code. Dependency analysis detects dependencies out of these traces i.e. execution traces are analyzed for identifying available dependencies. Dependency analysis output generates an intermediate representation in the form of dependence graph. Graph generation module is concerned about creation of graph for dependent components of code and storing this graph in the memory in an effective manner. The main aim here for analyzing dependencies is to provide parallel computation of code. So based on the analyzed output independent streams are executed in parallel which results in improved processing time of system.

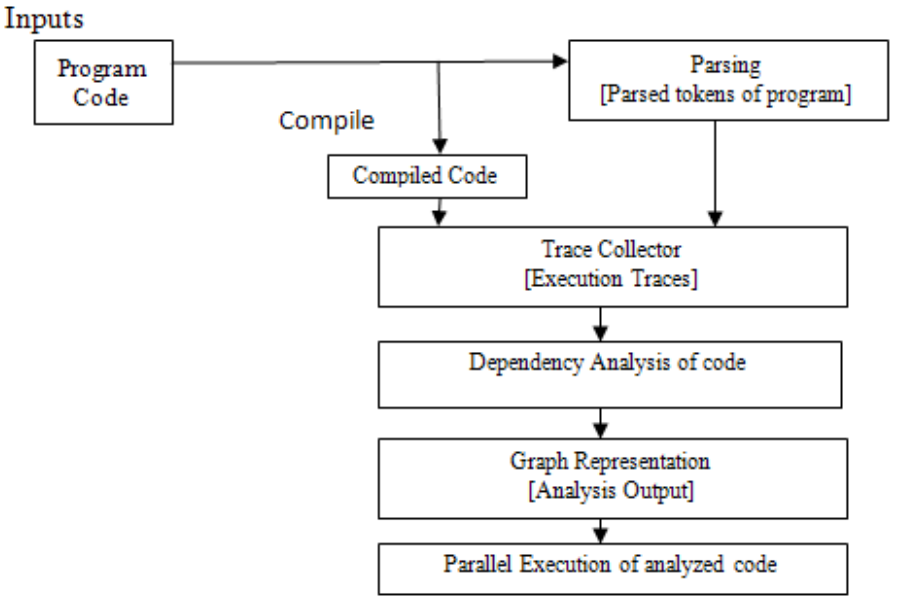

Fig 3. Proposed dependency analysis model

7. Discussion and conclusion

Dependency analysis is an important factor in various areas like parallel processing, high performance computing, in natural language processing, parallelizing compilers, for service oriented architecture and almost in all phases of software engineering. There are various dependency analysis approaches based on the type of input and level of abstraction considered for analyzing dependencies. All such approaches are discussed in paper along with their way of analysis. These methods have their own advantages and limitation when applied in practice. So this results in a need to develop a novel method that will give improved performance for automatic dependency identification and analysis.

This initial research indicates that dependency analysis representation using conceptual graph is a more general and effective approach for analyzing dependencies. Conceptual Graphs provide a powerful way to represent, characterize, and analyze dependencies between the entities in the system. Using Conceptual Graphs, proposal of easier way to model entities, compared to matrix method and directed acyclic graphs method at different levels of abstraction. This paper proposes a system; which provides a new approach for dependency analysis and results in the reduced dependent analysis time, low memory consumption and less cost sensitive.

References

1. Trosky B, Callo Arias, Pieter van der Spek and Paris Avgeriou. A practice-driven systematic review of dependency analysis solution. *Empir Software Eng , Springer Journal* , 2011, DOI 10.1007/s10664-011-9158-8.
2. Angeles Navarro, Francisco Corbera, Rafael Asenjo, Rosa Castillo, Emilio L. Zapata. A data dependence test based on the projection of paths over shape graphs. *Elsevier Journal of Parallel Distributed Computing,* 72(2012)1547-1564, DOI:10.1016/j.jpdc.2012.08.004.
3. Sudhakar Sah, Vinay G. Vaidya . New Approach to Speedup Dynamic Program Parallelization Analysis. *ACM Journal, Published in: international Journal of Software Innovation* , Volume 2 Issue 4, October 2014, 28-47, IGI Publishing Hershey, PA, USA, DOI: 10.4018/ijsi.2014100103.
4. He Zhang, Juan Li , Liming Zhu, Ross Jeffery , Yan Liu , Qing Wang and Mingshu Li. Investigating dependencies in software requirements for change propagation analysis. *Journal on Information and Software Technology*, 2013 Elsevier, http://dx.doi.org/10.1016/j.infsof.2013.07.001.
5. Yongzheng Wu and Jun Sun, Yang Liu, Jin Song Dong. Automatically Partition Software into Least Privilege Components using Dynamic Data Dependency Analysis. *IEEE International Conference, ASE* 2013, Palo Alto, USA 978-1-4799-0215-6
6. William W. Song, Aijaz Soomro, Yang Li, Qin Liu. A Structural Analysis of SLAs and Dependencies Using Conceptual Modelling Approach *37th Annual Computer Software and Applications Conference Workshops,* 2013 IEEE, DOI 10.1109/COMPSACW.2013.112
7. Mia Kamayani and Ayu Purwarianti. Dependency Parsing for Indonesian. *International Conference on Electrical Engineering and*

Informatics Bandung, Indonesia 17-19 July 2011
8. Bart Pieters,Charles-Frederik Hollemeersch, Jan De Cock, Peter Lambert, Rik Van de Walle. Data-parallel intra decoding for block-based image and video coding on massively parallel architectures. *SignalProcessing:ImageCommunication27(2012)220–237, 2012 Elsevier Journal doi*:10.1016/j.image.2012.01.001
9. Tadeusz Pankowski, Tomasz Pilka. Dealing with redundancies and dependencies in normalization of XML data. *Proceedings of the International Multi conference on Computer Science and Information Technology* pp. 543–550, 2008 IEEE
10. Calin Glitia, Pierre Boulet, Eric Lenormand, Michel Barreteau. *Repetitive model refactoring strategy for the design space exploration of intensive signal processing applications*. *Journal of Systems Architecture* 57 (2011) 815–829,doi:10.1016/j.sysarc.2010.12.002
11. Mihai T. Lazarescu and Luciano Lavagno. Dynamic trace-based data dependency analysis for parallelization of C programs. *12th International Working Conference on Source Code Analysis and Manipulation,* 2012 IEEE, DOI 10.1109/SCAM.2012.15
12. M. A. Hossain, U. Kabirb, M.O. Tokhi. Impact of data dependencies in real-time high performance computing. *journal on Microprocessors and Microsystems,* 2002 Elsevier Science
13. J. Zhao, R. Zhao, X Chen, Bo zhao. An improved nonlinear data dependence test. *Springer Journal of Supercomputing (*2015), DOI10.1007/s11227-014-1298-3
14. Sudhakar Sah and Vinay G. Vaidya. New Approach to Speedup Dynamic Program Parallelization Analysis. *International Journal of Software Innovation, December* 2014, DOI: 10.4018/ijsi.2014100103
15. Alain Ketterlin Philippe Clauss. Profiling Data-Dependence to Assist Parallelization: Framework, Scope, and Optimization. *IEEE/ACM45th Annual International Symposium on Microarchitecture*, 2012 IEEE DOI 10.1109/MICRO.2012.47
16. Judit Planas, Rosa M. Badia, Eduard Ayguadé and Jesus Labarta. HIERARCHICAL TASK-BASED PROGRAMMING WITH STARSS. *The International Journal of High Performance Computing Applications*, Volume 23, No. 3, 2009, pp.284-299, DOI: 10.1177/1094342009106195
17. John Ye, Hui Yan, Honglun Hou and Tianzhou Chen. Potential thread-level-parallelism exploration with superblock reordering. Springer-Verlag Wien 2014, DOI 10.1007/s00607-014-0387-8
18. Jeremy Villalobos and Barry Wilkinson. Using Hierarchical Dependency Data Flows to Enable Dynamic Scalability on Parallel Patterns. 2011 *IEEE International Parallel & Distributed Processing Symposium*, DOI 10.1109/IPDPS.2011.242.
19. Anne Benoit, Umit V. C Atalyurek, Yves Robert And Erik Saule,. A Survey of Pipelined Workflow Scheduling: Models and Algorithms. *ACM Computing Surveys,* Vol. 45, No. 4, Article 50, August 2013 , DOI: http://dx.doi.org/10.1145/2501654.2501664.
20. Paolo Baldan , Silvia Crafa, "A Logic for True Concurrency", Journal of the ACM (JACM), Volume 61 Issue 4, July 2014, ACM New York, NY, USA, doi>10.1145/2629638.
21. Christoph Niethammer, Colin W. Glass and Jos´e Gracia. Avoiding Serialization Effects in Data/Dependency aware Task Parallel Algorithms for Spatial Decomposition. *10th IEEE International Symposium on Parallel and Distributed Processing with Applications,* 2012 IEEE,DOI 10.1109/ISPA.2012.109
22. Lisa Cox, Dr. Harry S. Delugach, Dependency Analysis Using Conceptual Graphs. http://citeseerx.ist.psu.edu, doi=10.1.1.8.1136
23. Gabriele Bavota, Malcom Gethers, Rocco Oliveto, Denys Poshyvanyk And Andrea De Lucia. Improving Software Modularization via Automated Analysis of Latent Topics and Dependencies,. *ACM Transactions on Software Engineering and Methodology*, Vol. 23, No. 1, Article 4, February 2014, DOI: http://dx.doi.org/10.1145/2559935.
24. Arun Sharma, P. S. Grover and Rajesh Kumar. Dependency Analysis for Component-Based Software Systems. *SIGSOFT Software Engineering* Notes, July 2009 Volume 34 Number 4, DOI: 10.1145/1543405.1543424.