REPORTS AND STUDIES IN
FORESTRY AND NATURAL SCIENCES

Martti Penttonen (ed.)

Computer Science I Like

Proceeding of Miniconference 4.11.2011

Publications of the University of Eastern Finland
Reports and Studies in Forestry and Natural Sciences

UNIVERSITY OF
EASTERN FINLAND

MARTTI PENTTONEN (EDITOR)

Computer Science I Like

Proceedings of Miniconference on 4.11.2011

Publications of the University of Eastern Finland
Reports in Forestry and Natural Sciences
No 6

Kopijyvä
Kuopio, 2011
Editor: Prof. Pertti Pasanen

Distribution:
University of Eastern Finland Library / Sales of publications
P.O. Box 107, FI-80101 Joensuu, Finland
http://www.uef.fi/kirjasto

ISBN: 978-952-61-0635-9 (printed)
ISSNL: 1798-5684
ISSN: 1798-5684
ISBN: 978-952-61-0636-6 (pdf)
ISSN: 1798-5692 (pdf)

Preface

This proceedings is outcome of a miniconference held on 4.11.2011, at the University of Eastern Finland. The editor of this book, before retirement a month later, invited his former PhD students to give a talk under title *Computer Science I Like*. The point behind the title was that scientific research is not just work but also a vocation. As an event, the miniconference was a more or less regular scientific conference, although only one day long and with a wider scope. For students it offered a window into the world of research. Also, it was fun to meet. I thank all those, who made this miniconference and proceedings possible.

Kuopio, 5 December, 2011 Martti Penttonen

Contents

Why do we need software engineering?

Anne Eerola
School of Computing
University of Eastern Finland
P.O. Box 1627, 70211 Kuopio, Finland

Abstract. The background of the modern software engineering is based on the software crisis. It was noticed that the quality of software is not as good as it should be. Since then, we have got a lot of methods and tools, but we still have challenges to build up high quality software that can be maintained, too. Difficulties occur especially in developing large, heterogeneous, and distributed information systems. In this paper, we emphasize some of the most important concerns of software engineering. We cover a path from requirements engineering, through architecture development and testing, to component-based software. We highlight the methods and practices which promote the production of software fulfilling the requirements of business and stakeholders. Additionally, we propose that IT Service Management is an important part of software engineering.

1 INTRODUCTION

Information systems are nowadays complex, distributed, and they need to be changed and maintained according business requirements. Additionally, many information systems, for example, in health care and banking domains are critical and require high quality of software and its production process. In this paper, we consider why software engineering is needed and how does it assist the development of information systems.

The background of the modern software engineering is based on Year 1968 as an answer to the software crisis. It was noticed that it is difficult to write correct and understandable computer programs, which take into consideration change management, too. *Software Engineering* (SE) is an engineering discipline which focuses on cost effective development of high-quality software systems [32]. While computing concentrates on computers, programming languages and algorithms SE emphasizes tools to solve problems in software industry or customer organizations [28].

Software Engineering investigates software production (requirements engineering, analysis, design, implementation, software testing and inspection, deployment, and maintenance). Moreover, quality management, product management, and the structure of software systems, i.e. architectures, components, frameworks, and patterns are considered [9, 32]. The emphasis is mainly on technical aspects of software systems. However, most software engineers agree that it is not possible to start a software development project without considering carefully the requirements of the stakeholders. After deployment, customers and users need support in using software systems. Hence, we develop methods, techniques, and processes of IT service management and propose that IT service management is an important part of SE [13, 14].

Intelligent thinking requires that a designer focuses his/her attention upon some aspect and follows the *separation of concerns principle*: "We know that a program must be correct and we can study it from that viewpoint only; we also know that it should be efficient and we can study its efficiency on another day" [4]. *Concerns* can be responsibilities, functions, data, features, tasks, qualities, transactions, services, or any aspect of the requirements or design [3,28].

Modularization is concerned with the meaningful decomposition of the software system and with its grouping into subsystems, components, and services [3, 24]. Since structured design [40] emerged, high cohesion and low coupling have been attributed to indicate high quality of software. The term module can be defined as "a syntactical or conceptual entity of a software system" [3]. A mod-

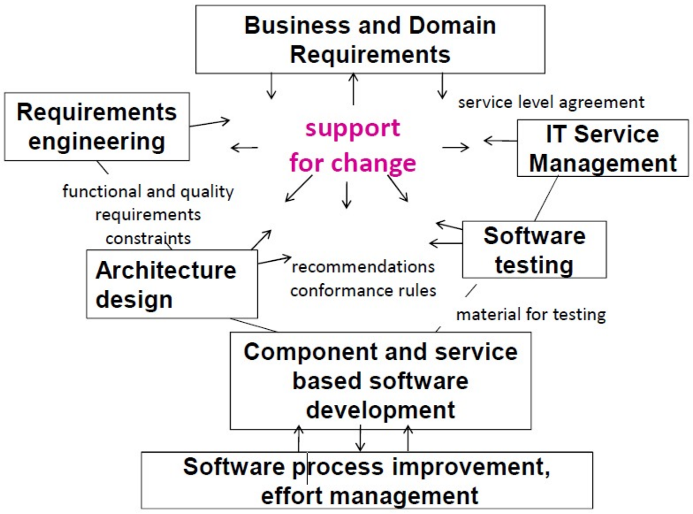

Business and Domain Requirements
Requirements engineering
support for change
service level agreement
IT Service Management
functional and quality requirements constraints
Software testing
Architecture design
recommendations conformance rules
material for testing
Component and service based software development
Software process improvement, effort management

Figure 1: Research in Software Engineering

ule is a component, a class inside a component, or a service using which the functionality is offered to the customer.

Research in software engineering at the University of Kuopio (now University of Eastern Finland) started in 1990s. In our research group requirements engineering, architecture design, software testing, IT service management, and effort management are emphasized (see Fig. 1). Typical research methods are case studies, action research, and constructive research. Data collection is based on multiple sources of evidence (field visits, workshops, participative observation in meetings, interviews, access to organizations tool applications, and internal documentation). The research results can involve, for example, identification of process challenges; creation of new methods, and models; providing process implementation guidelines and recommendations; as well as improving data, quality, and service management. The group belongs to Information Systems and Software Engineering research group.

In this paper the research question is: Why do we need software engineering? We aim to answer the question by highlighting research results that from our opinion are typical and important to software engineering. Additionally we give an overview of our research results. The rest of this paper is organized as follows: Chapter 2 considers effort management as a basis of software process management. In Chapter 3, we consider requirements engineering as a collection of stakeholders wants and needs. Chapter 4 discusses software architecture development, especially component based software development and integration. Chapter 5 emphasizes software testing and its importance. Chapter 6 introduces IT service management. Chapter 7 concludes the paper.

2 EFFORT MANAGEMENT

Usually a software project starts with feasibility study and effort estimation. Accurate effort estimation is important because it enables to keep project schedule, budget, and time-to-market calculations. Successful resource allocation decreases working pressure and haste of software engineers, too. Haapio proposed that non-construction activities should be taken into account in addition to the constructive and project activities [7]. Non-constructive activities are, for example, configuration management, customer-related activities, documentation, orientation, and quality management. Effort management continues through the whole software development project and we should help software engineers to adopt general project activities in order to increase reliability of registered effort in time-booking entries.

Haapio et al. [8] proposed a novel stepwise method for assessing software project effort. The method provides necessary tools for the effort management and assessment. When utilized for post-mortem analysis, the method can be used for deciding on the project result and how the project differs from other projects (quantity, quality, and effort views).

Virtanen [38] has considered effort estimation in component based software development. He has strived to increase reusability and productivity in software development. The starting points in

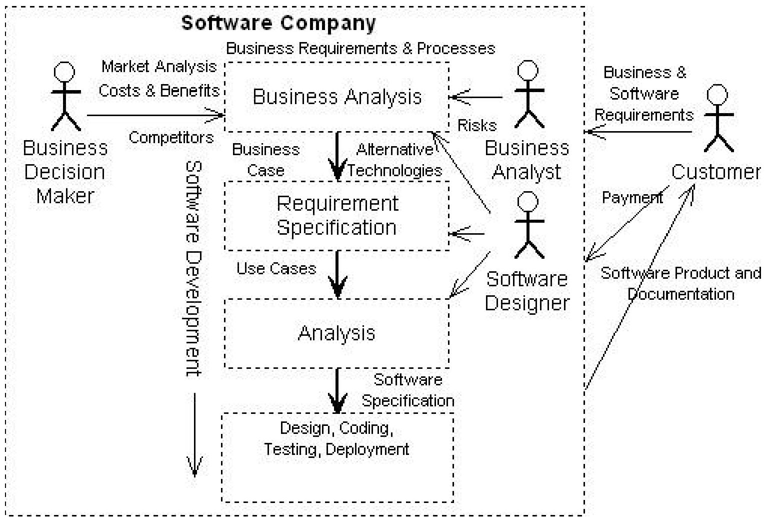


Figure 2: Business case as a starting point for requirements engineering [15]

method are product structure and history data of projects. The average effort of components is after calculation corrected emphasizing project and human effects.

3 REQUIREMENTS ENGINEERING

The gap, between business decision making and software engineering, causes inefficiency and quality problems in software development. Software engineers do not understand organization's value creation objectives and their influence on software production and structure. For this reason, software does not fulfil business requirements and software quality is inadequate too often. Hence, we utilize a business case as a starting point for requirements engineering (see Fig. 2).

Software systems must fulfill the requirements of the stakeholders. Stakeholders are people, who have interest in the product or who will be affected by products use [30]. "A requirement is a condition or capability of the software or its component needed by a user to solve a problem or achieve an objective" [11]. A requirements specification describes what the system must do and which properties the system must have to satisfy a contract, a standard, or other user requirements [11, 30]. Requirements errors are the most expensive and the most dangerous software errors [21]. It is important to gather, analyze, validate, and master the requirements carefully.

Functional requirements define functions of a software system, or its component and services, that the system must provide [32]. Additionally, functional requirements are data requirements (i.e. the system must store, maintain, and managed data) and things that the system is not allowed to do [30]. These requirements are currently most often presented in the form of *use cases* described in the almost standard Unified Modeling Language UML [5].

Quality requirements of software are product, organizational, and external requirements, for example, usability, standardization, and safety [32]. One component of the software needs not to be as high quality as another, if the components do not depend on each other and if there is no side-effect. Quite often, quality attributes of software cannot be pinpointed to a certain component [2]. Prioritization and definition of *sensitivity points* for quality attributes is needed [17]. *Quality requirements of a production process* are qualities that software engineers appreciate, for example, maintainability and reusability [2]. As before, one step of the production process needs not to achieve as high quality as another, if the importance and risks of steps differ. A research problem is, how we can find the components and production steps that are most important to be emphasized.

We strive to improve requirements engineering (RE) processes and models through following ways:

- We have derived guidelines for managing requirement process in case of large information systems and different professionals (e.g. nurses, doctors, software engineers, researchers).

- We keep in mind that software requirements should be described in such a way that architecture and forthcoming software are convenient to design (although the process may stop, if it turns out after the RE-process that software is not a right solution to current problems).

- We have generated understandable models for requirements specification contributing architecture design decisions and detailed requirements specifications (e.g. see Fig. 3).

One problem is that we need a common language for all the stakeholders in the RE-process and forthcoming software development project. Another great problem in RE is that information systems development processes of customer organizations and software providers do not meet each other. Hence, the interaction between the software provider and its actual client does not occur in the software providers RE phase. It may happen the product is built before its requirements are specified [22]. Component based software development, generic products, and parameter utilization assist this problem, but do not solve it thoroughly.

4 SOFTWARE ARCHITECTURE DEVELOPMENT

4.1 Software architecture, architectural styles, and patterns

Architectures and design models are important tools to assure software quality and to improve software reuse. IEEE Standard 1471-2000 defines the *software architecture* as "The fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution". The software architecture consists of statements describing how a system is decomposed into its component parts and how the functionality is allocated among those parts. Smolander [31] has considered the role of architecture in systems development in several organizations. For example, the usage of architectural viewpoints and the rationale of architecture design and description are considered.

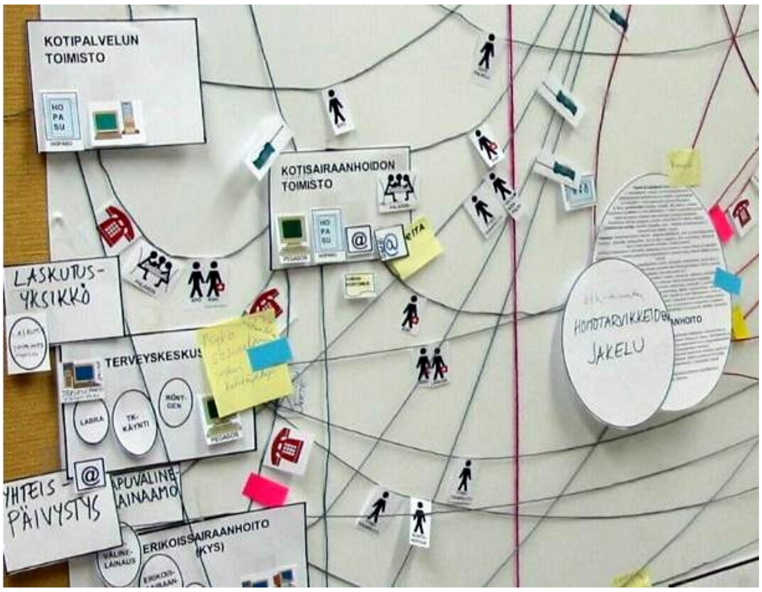


Figure 3: Home Care -case [34]

Architecture analysis and design offer means to understand business goals and stakeholder concerns and map these onto an architectural representation, and assess risks associated with this mapping [17]. The architecture is a blueprint and an implicit high-level plan for software construction [19].

Software engineering team can use architectural styles, first proposed by Perry and Wolf [27]. The *architectural style* defines types of components and connectors (i.e. building blocks), constraints for using the building blocks, vocabulary, and analysis to reason critical and risky properties of the solution [2,3]. Examples of architectural styles are Pipes and Filters, Broker, Repository, and Layered styles.

In addition to styles, software engineers utilize patterns, for example, architectural, design, usability, security, and organizational patterns [3, 6]. The described "*pattern* is a general and proven solution to a frequently occurring architecture or design problem in a context" [1, 6, 18]. Examples of patterns are Proxy, State, Facade,

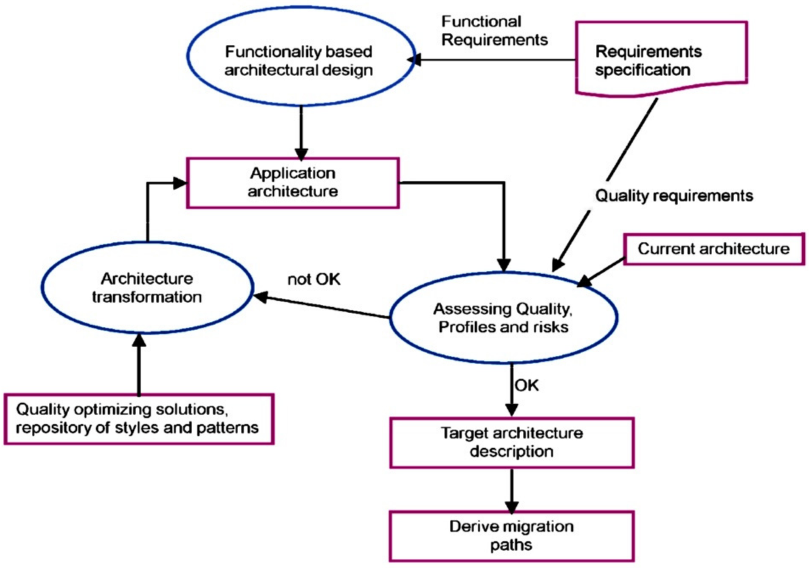

Functionality based architectural design
Functional Requirements
Requirements specification
Application architecture
Quality requirements
Current architecture
Architecture transformation
not OK
Assessing Quality, Profiles and risks
OK
Quality optimizing solutions, repository of styles and patterns
Target architecture description
Derive migration paths

Figure 4: Target architecture design

Role, and Undo.

The functionality of the component based system can be offered to the customers using service oriented architecture. This supports generation of modifications. "A Service-Oriented Architecture (SOA) is an architecture that is based on the key concepts of a user interface, service, service repository, and service bus." [19].

The *target architecture* defines the vision where we want to develop our information system. Usually it is not possible to move to the target state at once but we need a *migration path* from the current legacy system to the target architecture [25]. Architecture design, assessment, and evolution are described in Fig. 4.

4.2 Component and service based software development

The structure of software is defined using components and services. Component based software development (CBSD) and service oriented architectures (SOA) have emerged perhaps to the most popular software development approach today: In CBSD software is constructed using self-made and commercial off-the-shelf (COTS) components and glue code if needed. The idea is not new, but McIlroy [23] proposed mass-production of components already in 1969. A *software component* is a binary unit for composition according to the contract and explicit dependencies [33]. It is self-contained piece of software, which encapsulates a set of related functions and data and can be independently deployed and plugged into a compatible environment [10]. At run-time the component is accessed through a set of interfaces [32]. Similarly build-time interfaces (proxy export) are advantageous to be utilized in order to shorten time-to-market [10]. A component has no persistent state. The relationships and interfaces between components and the internal structure of each component can be described with UML Diagrams and XML document type definitions (DTD) [20]. While designing component based systems we should strive to low coupling and high cohesion (see Fig. 5).

Component based systems are often object oriented. In object oriented approach, things in the real world are abstracted as objects, which encapsulate attributes and methods. Objects communicate with each other by sending messages. These objects can be related to each other by aggregation, by association, and by classification. Objects are defined in a class. Classes, in turn, form a class hierarchy, in which subclasses inherit properties, i.e. attributes and methods from super classes [5,29].

A *service* fulfils a contract and it has one or more interfaces and an encapsulated implementation [19]. Each service is a software component of distinctive functional meaning (business logic and data) and is found from a repository. Services are typically not linked as code libraries but are bound to at runtime.

Our goal has been to develop methods and tools for service-oriented software development emphasizing business aspects and quality of software structure (see Fig. 6). We propose a service map for service workflow definitions, utilize Service Level Agreement (SLA) based on IT Service management framework [12], and

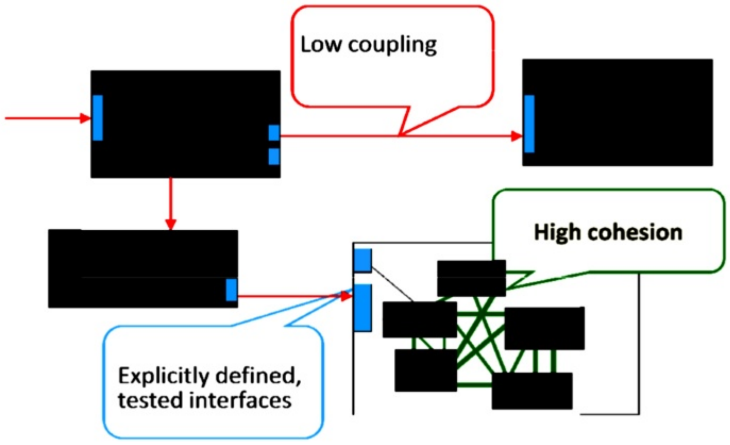


Figure 5: Low coupling, high cohesion, and explicitly defined and tested interfaces

clarify protocol requirements using distributed coordination in the SOA environment. In addition, software service identification and design has been studied.

4.3 Integration

The focus of the software development process has moved from separate and independent applications to the integrated and distributed software systems. Vänttinen [39] has strived to provide quicker and more effective way, with smaller resources to launch software products to the markets, and to enable the integration between software systems, provided by different vendors.

His thesis introduces enterprise application integration (EAI) and service oriented architectures (SOA) and analyzes three real-life case studies to find out how principles of EAI, SOA, and software product families can be used efficiently in software industry.

Acquired components and legacy systems are integrated according to the target architecture [26] in order to get the software system, which fulfils the requirements of stakeholders. Integration

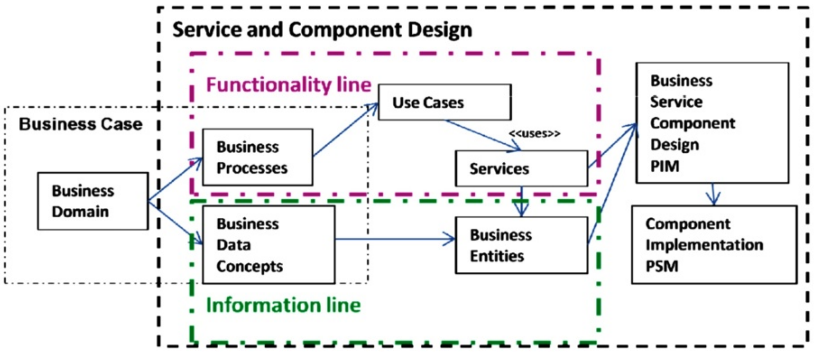


Figure 6: Framework for service oriented software development [15]

of software pieces can be done by third party using interface definition language (IDL or XML) and tested interfaces [33]. Additionally, conformance to the standards is advantageous to be tested [36].

Mykkänen [25, 26] has considered health information systems. He has proposed methods, models, and guidelines for developing reusable integration solutions using a component-based and model-driven approach:

- Methods to improve interoperability of software system
- A generic integration process with integration specification levels that support collaborative integration definition,
- A method for stepwise migration from current legacy architecture to component-based target architecture, and
- Experiences and recommendations for integration based on several pilot projects in software industry and healthcare/welfare service providers.

5 SOFTWARE TESTING

Nowadays software systems are not built in green field but software is constructed from ready-made COTS (commercial off the shelf) which must cooperate with each other. Hence, interoperability between applications is important. High quality and reliability requirements of software systems cause that software testing is important and testing techniques should be practical and easy to use. However, in network of organizations it is often impossible to test all side-effects when a change is accomplished in requirement specifications or code.

We have strived to improve software testing policy and methods. Components of different granularities were tested level by level leading to integration and conformance testing (see Fig. 7). We defined test cases based on UML diagrams and contracts of components. The dependency graph was used to assure that the whole functionality of the system has been covered. Usually, the internal logic of components is object-oriented and component systems are distributed.

Toroi [35,37] has proposed methods to help software companies to apply testing theory efficiently in practice and to improve interoperability by standards and conformance testing. The research questions were: How can we be sure that the product or an interface conforms to standards? How the recommendations, standards, and interface specifications should be defined, when we strive for testability? We considered three viewpoints: integrator, customer, and service provider.

Additionally, Toroi has considered test process improvement in industry and given a list of recommendations for improving test processes. She proposed that organizations have to pay more attention to the following issues: test case documentation, regression testing, the level of training in testing, clarity of specifications, and guidance of the test process improvement models.

6 IT SERVICE MANAGEMENT

In software industry there is a strong business need for IT service management (ITSM) research. Thousands of IT organizations worldwide have started to implement ITSM processes and organi-

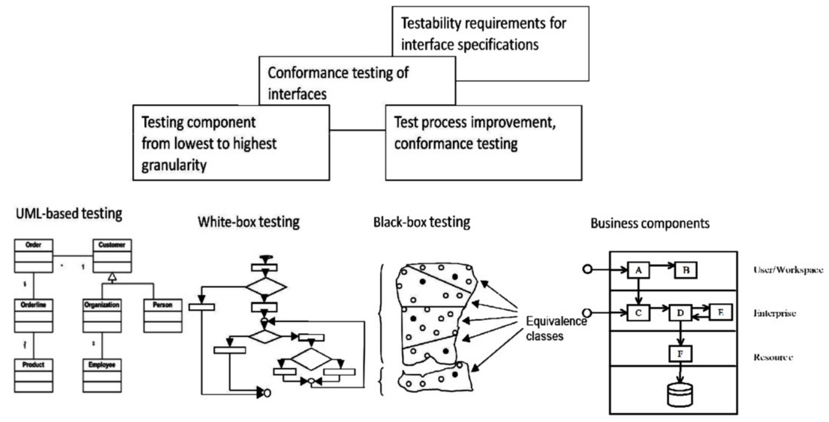

Testability requirements for interface specifications
Conformance testing of interfaces
Testing component from lowest to highest granularity
Test process improvement, conformance testing
UML-based testing
Order
Customer
Orderline
Organization
Person
Product
Employee
White-box testing
Black-box testing
Equivalence classes
Business components
A
B
C
D
E
F
User/Workspace
Enterprise
Resource

Figure 7: Testing Component-Based Systems

zations face several challenges while implementing processes. Jäntti [13] has considered difficulties in managing software problems and defects, which number has increased due to complex IT systems, new technologies, and tight project schedules. Regarding emerging new research topics it is common that the concepts are undefined, inconsistent, and inaccurate. Hence, conceptual model for IT Service Problem Management was proposed (see Fig. 8).

In our research projects MaISSI (Managing IT Services and Service Implementation) and KISMET (Keys to IT Service Transition) we have derived recommendations and guidelines for IT services and ITSM processes. Our viewpoints have been on designing, implementation, maintaining, and improvement. We have examined service implementation technologies and tools in close cooperation with industrial partners, i.e. IT service provider companies, software companies, and IT customer organizations. We have helped industrial organizations to identify challenges in their current activities. The goal has been to share knowledge on ITSM standards and frameworks.

The research validation is carried out in pilot projects that aim to solve problems that industrial partners have met in their service

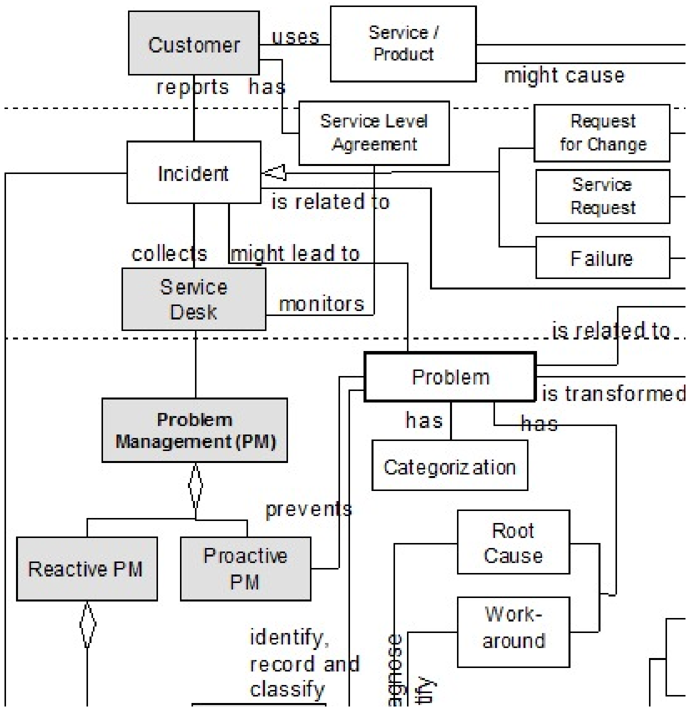


Figure 8: A part in a conceptual model of IT Service Problem management [14]

management business. For example, we have developed process descriptions for service support processes (incident management, problem management) and service transition processes (change management, configuration management, and continual service improvement). We have helped organizations in configuring service desk tools, examined how to measure the performance of support processes, and organized research workshops and seminars regarding ITSM. Industrial partners have utilized research results to increase the quality of ITSM processes, to identify the challenges in their current activities, to improve the performance and usability of the tools, and to enable effective knowledge sharing on service management standards and frameworks.

7 CONCLUSIONS

In this paper, we have emphasized research results that we propose are important for software engineers in research and industry. We have described the path from requirements engineering, through architecture design and software testing to software products and services. The presented research results form a toolbox needed to build on time and cost high quality software that fulfills stakeholders requirements. After deployment, software needs to be changed and evolved according to business requirements. Hence, IT service management is important.

References

[1] Alexander C., Ishikawa S., Silverstein M., Jacobson M., Fiksdahl-King I., and Angel S., A Pattern Language, Oxford University Press, New York, 1977

[2] Bosch J.: Design and use of software architectures, Addison-Wesley, 2000

[3] Buschmann F., Meunier R., Rohnert H., Sommerland P., and Stal M., Pattern-Oriented Software Architecture A System of Patterns, John Wiley & Sons, 2001

[4] Dijkstra, E., W., "On the role of scientific though, 1974" In Dijkstra, E. W. Selected Writings on Computing: A Personal Perspective. New Yourk: Springer-Verlag, 1982

[5] Fowler M.: UML Distilled A Brief Guide to the Standard Object Modeling Language, Addison-Wesley, 2004

[6] Gamma E., Helm R., Johnson R., and Vlissides J., Design Patterns Elements of reusable Object-Oriented Software, Addison-Wesley, 1995

[7] Haapio Topi: Improving Effort Management in Software Development Projects, Doctoral dissertation, University of Eastern Finland, 2011

[8] Haapio T., Eerola A.: Software Project Effort Assessment. Journal of Software Maintenance and Evolution: Research and Practice. 22(8), pp 629-652, 2010.

[9] Haikala I., Märijärvi J., Ohjelmistotuotanto, 1998, 2004,

[10] Herzum P., Sims O.: Business Component Factory A Comprehensive Overview of Component-Based Development for the Enterprise, Wiley Computer Publishing, 2000

[11] IEEE. IEEE Standard Glossary of Software Engineering Terminology. New York: IEEE, 1983. Leite, J. Viewpoint Resolution in Requirements ANSI/IEEE Std. 729-1983.

[12] IT Infrastructure Library, Office of Government Commerce (OGC), 2002 Continual Service Improvement, OGC 2007

[13] Jäntti Marko: Difficulties in Managing Software Problems and Defects, Doctoral dissertation, Department of Computer Science, University of Kuopio, 2008.

[14] Jäntti M., Eerola A.; A Conceptual Model of IT Service Problem Management, Proceedings of the IEEE International Conference on Service Systems and Service Management ICSSM06, 2006

[15] Karhunen H., Jäntti M., Eerola A.: Service-Oriented Software Engineering (SOSE) Framework, International Conference on Service Systems and Services Management 2005, Proceedings ICSSSM05, 2005

[16] Kazman R., The Essential Components of Software Architecture Design and Analysis, Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC05), 2005

[17] Kazman R., Bass L., Abowd G., Webb M., SAAM: A Method for Analyzing the Properties of Software Architectures http://www.sei.cmu.edu/library/assets/ICSE16.pdf, referred 11.4.2011

[18] Koskimies K., Mikkonen T.: Ohjelmistoarkkitehtuurit, Talentum Media Oy, 2005

[19] Krafzig D., Banke K., and Slama D., Enterprise SOA Service-Oriented Architecture Best Practices, Prentice Hall, Pearson Education, 2005

[20] Kuikka E., Eerola A., A Correspondence between UML Diagrams and SGML/XML DTDs, Lecture Notes in Computer Science, Springer-Verlag, 2023/2004

[21] Axel van Lamsweerde: Requirements Engineering, John Wiley Sons Ltd, 2009

[22] Luukkonen I., Eerola A.: Improving requirements Engineering from the clients perspective in the health care domain, Proceedings of the 25th conference on IASTED International Multiconference: Software Engineering, 2007

[23] McIlroy M., D., Mass produced software components. In Naur P. and Randell B., Software Engineering, Report on a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968, Scientific Affairs Division, NATO, Brussels, 1969,138-155.

[24] Meyer B., Object-Oriented Software Construction, Prentice Hall International, 1988

[25] Mykkänen Juha: Specification of Reusable Integration in Health Information Systems, Doctoral dissertation, Department of Computer Science, University of Kuopio 2007

[26] Mykkänen J., Tikkanen T., Rannanheimo J., Eerola A., Korpela M.: Specification Levels and Collaborative Definition for the Integration of Health Information Systems, In Baud R, Fieschi M, Le Beux P, Ruch P, eds. The New Navigators: from Professionals to Patients, Proceedings of MIE2003, Saint-Malo, France, 4-7 May 2003

[27] Perry D.E. and Wolf A.L., Foundations for the Study of Software Architecture, ACM Software Engineering Notes, 17, 4,October 1992, 40-52

[28] Pfleger S.,L., and Atlee J.M., Software Engineering Theory and Practice, Pearson Education, Prentice Hall, 2010

[29] Putkonen A.: A Methodology for Supporting Analysis, Design and Maintenance of Object-oriented Systems, Academic Dissertation, Department of Computer Science and Applied Mathematics, University of Kuopio, 1994

[30] Robertson S., Robertson J.: Mastering the Requirements Process, Addison-Wesley, 1999

[31] Smolander Kari: On the Role of Architecture in Systems Development, Department of Information Technology, Lappeenranta University of Technology, 2003

[32] Sommerville I., Software Engineering, Eight Edition, PearsonEducation, 2011 Software Architecture Definitions, http://www.sei.cmu.edu/architecture/start/community.cfm and IEEE Standard 1471-2000

[33] Szyperski C., Component Software Beyond Object-Oriented Programming, Addison-Wesley, 1999

[34] Toivanen M.: Home Care -case [PlugIT-hanke], 2004

[35] Toroi T.: Testing Component-Based Systems Towards Conformance Testing and Better Interoperability, Doctoral dissertation, Department of Computer Science, University of Kuopio, 2009.

[36] Toroi T., Eerola A., Mykkänen J., Conformance Testing of Interoperability in Health Information Systems in Finland. In: Kuhn K., Warren J., Leong T-Y, eds. Medinfo 2007, Brisbane, Australia, August 20-24, 2007, p. 127-131. Amsterdam: IOS Press, 2007.

[37] Toroi T., Eerola A.: Requirements for the Testable Specifications and Test Case Derivation in Conformance Testing, In: Dasso A, Funes A, eds. Verification, Validation and Testing in Software Engineering, P. 118-135. Hershey: Idea Group Publishing, 2006

[38] Virtanen, P., 2003. Measuring and Improving Component-Based Software Development. Doctoral dissertation, Finland: University of Turku.

[39] Vänttinen Pasi: Integrating Applications in Software Company, Licentiates thesis, Department of Computer Science, University of Kuopio, 2008

[40] Yourdon E., and Constantine L, Structured Design, Fundamentals of a Discipline of Computer Program and Systems Design, Englewood Cliff, NJ: Prentice Hall, 1978

Computer Architecture Re(de)fined —the Era of Parallel Computing

Martti Forsell[1]
[1] VTT, Computing Platforms, Box 1100, FI-90571 Oulu, Finland
Martti.Forsell@VTT.Fi
http://www.ee.oulu.fi/~mforsell/work.html

Abstract. Now that all the major general purpose processor manufacturers have irreversibly switched to multicore processor architectures with aims to increase the number of processor cores per processor chip as fast as evolving silicon technologies makes it possible to pack more transistors on them, it is time to take a look at the architectural ideas behind them. A simple classification of architectural approaches in current multicores reveals that the symmetric multiprocessor and non-uniform memory access, which lend most of their performance enhancement techniques from sequential computers, are by far the two most common ones. Since our measurements and also wide-spread consensus indicate that the performance and programmability of these approaches is far from perfect, we ask wether these techniques suit best to multicore style thread-level parallel execution or are there possibly other/new techniques that would provide better performance. Based on our on-going analysis of parallel execution techniques, we are tempted to answer negatively to this question. More specificly, we claim that techniques, like multithreading assisted high-throughput computing providing scalable latency hiding, chained static superscalar execution allowing easy exploitation of low-level virtual instruction-level parallelism, combining/active memory techniques enabling concurrent memory access and multioperations, and wave based light-weight implicit synchronization making possible to emulate strong synchronous models of computation, provide much higher gains in performance and programmability especially for general purpose computation than current ones. Even though this is partially based on preliminary models and on-going work we believe that they already point relatively well to the right direction. Detailed and verified analytic performance models of TLP approaches and thorough discussion on implications will be published in the later phases of this research.

1. Introduction

Computer architecture is a branch of science focusing on the theory and practice of designing, selecting and interconnecting hardware components to create computers that meet functional, performance and cost goals and the formal modelling of those systems [Hennessy03]. An abstract (sequential) computer consists of a processor that is connected to the memory and input/output data. The processor is able to read and execute instructions one by one from the memory. While executed, instructions read one or more input data or memory location values, do a simple computation for this data, and write the result to output data or memory. A program is a sequence of instructions that alters the state of the memory so that the output data (results) can be obtained from the input data. A realization of a computer consists of these basic components typically manufactured from semiconductor elements, like transistors, wires, insulators, resistors, and condensators that in turn form basic building blocks of digital systems known as logic gates, memory cells, and registers. These blocks will then be used to build up a processor, memory and input/output devices. As a physical device, this kind of a computer takes always some space (or area on silicon), consumes some power, and is able to execute instructions with some bounded rate. The laws of physics and properties of the used (silicon) manufacturing technology set practical limits to the performance of the processor and size and speed of the memory as well as to the bandwidth of input/output data. The first computers were build according to this kind of Von Neumann architecture, where both the program and data resided on the same memory so that instruction execution involving fetch of instruction and actual execution can not happen in parallel (the Von Neumann bottleneck). The key techniques to improve the performance of this kind of a basic execution engine connected to the memory and input/output devices include the Harward architecture, pipelining, dynamic superscalar execution, caching, and various speculations. The Harward architecture removes the Von Neumann bottleneck by separating the data and instruction memory accesses with a help of dedicated paths. Pipelining [Bloch59, Bucholz62] and dynamic superscalar execution [Thornton64, Anderson67, Tomasulo67] improve the throughput of instruction execution and exploitation of available *instruction-level parallelism* (ILP). They make use of various speculations [Johnson89, Hennessy03] to reduce the delays due to memory access, control transfer, and pipelining. Caching [Kilburn62] is used for hiding the latency of the memory systems and balancing the speed difference between processor and memory devices.

In the early 2000's, the clock frequency development of processors and memories almost halted after being exponential for decades since the power density of ever tightly packed semiconductor elements reached its upper limit [Mazke97, Brooks00, Mudge01, Intel06, ITRS10]. As a results, all major processor manufacturers switched irreversibly to multicore processor architectures and thus to the *thread-level parallel* (TLP) computing paradigm with aims to increase the number of cores per processor chip as fast as the silicon technology makes it possible to pack more transistors on a single chip. The manufacturers naturally hope that by increasing the number of cores, users of multicore processor based computers get respectively more performance.

Currently by far the two most popular architectural approaches for multicore computing are *symmetric multiprocessor* (SMP) and *non-uniform memory access* (NUMA) both making use of the shared memory intercommunication scheme but lending most of their performance enhancement techniques from sequential computers. Since our measurements [Forsell02, Forsell11a, Forsell11c] and also wide-spread consensus among parallel computer experts indicate that the performance and programmability of these approaches is far from perfect, we ask wether these techniques suit best to multicore style thread-level parallel execution or are there possibly other/new techniques that would provide better performance. Based on our on-going analysis of parallel execution techniques, we are tempted to answer negatively to this question and claim that techniques, like multithreading assisted high-throughput computing providing scalable latency hiding, chained static superscalar execution allowing easy exploitation of low-level virtual instruction-level parallelism, combining/active memory techniques enabling concurrent memory access and multioperations, and wave based light-weight implicit synchronization making possible to emulate strong synchronous models of computation, provide much higher gains in performance and programmability especially for general purpose computation. Even though this is based on preliminary models and on-going work we believe that they point already to the right direction. Detailed and verified analytic performance models of TLP approaches and thorough discussion on implications will be published in the later phases of this research.

The rest of the paper is organized so that in Section 2 we will take a look at existing TLP architectures, in the Sections 3-7 we discuss performance enhancement techniques for latency hiding, synchronization, virtual ILP exploitation, concurrent memory access and multioperations respectively, and finally, in Section 8 we give our conclusions.

2. Existing approaches to thread-level parallel execution

The main approaches to TLP computing, classified according to the used intercommunication mechanism, are message passing and shared memory. In the message passing approach processor cores have local memories only and computational subtask communicate by sending messages to each other. The dataflow computing can be seen as a variant of this approach. In the shared memory approach cores are connected to either unified or distributed shared memory and they communicate via variables on it. Since virtually all current general purpose multicore processors provide a shared memory abstraction, we focus here on the two most popular variants of it—symmetric multiprocessor and non-uniform memory access.

Symmetric multiprocessor (SMP) is a simple architectural approach for thread-level parallel computing that is being used typically for a small number of processor cores. An SMP consists of a number of identical (often superscalar) processors with local caches, which are connected to the main memory via a bus (unified case) or crossbar (distributed case) (see Fig. 1). All memory locations are equidistant to all processors, thus access is symmetric. Caches are used in between the processors and the main memory and they are kept coherent using a bus snooping mechanism [Culler99, Hennessy03].

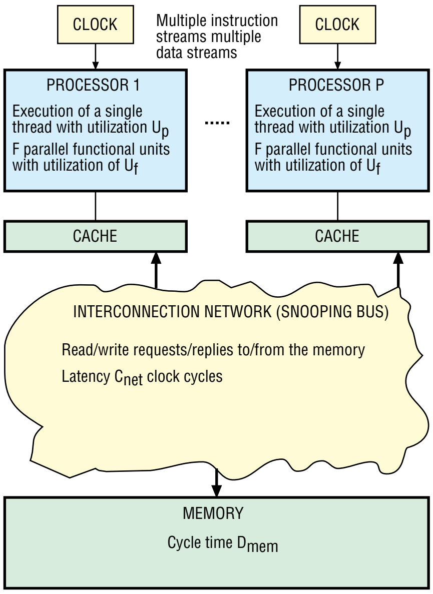

CLOCK
Multiple instruction streams multiple data streams
CLOCK
PROCESSOR 1
Execution of a single thread with utilization U_p
F parallel functional units with utilization of U_f
.....
PROCESSOR P
Execution of a single thread with utilization U_p
F parallel functional units with utilization of U_f
CACHE
CACHE
INTERCONNECTION NETWORK (SNOOPING BUS)
Read/write requests/replies to/from the memory
Latency C_{net} clock cycles
MEMORY
Cycle time D_{mem}

Fig. 1. Block diagram of the SMP approach. (P=number of processors, F=number of functional units, C_{net}=latency of the intercommunication network, D_{mem}=cycle time of the memory, U_p=utilization of processors, U_f=utilization of functional units.)

Non-uniform memory access (NUMA) is a distributed shared memory approach in which multiple identical (often superscalar) processors with their local caches and memory banks are connected together via an intercommunication network (memory is distributed, only local memory is cached, see Fig. 2). Non-local memory accesses have higher (distance and traffic situation dependent) latency than local accesses [Swan77]. It is believed to provide a more realistic memory access concept than SMP— now the fact that distant memory accesses take longer than local ones is projected at the architectural approach level— making it much easier to retain the clock rate of the design fixed as the number of processor cores increases.

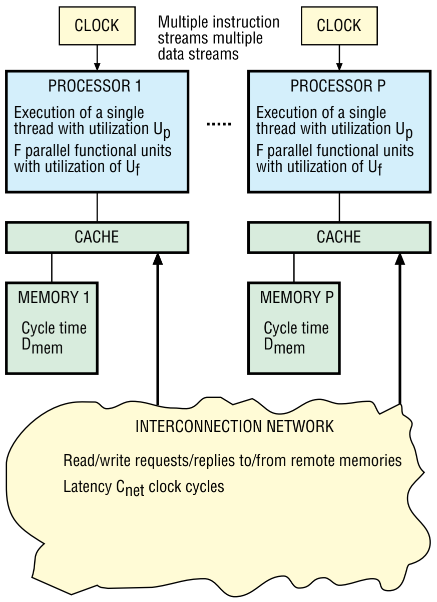

CLOCK
Multiple instruction streams multiple data streams
CLOCK
PROCESSOR 1
Execution of a single thread with utilization Up
F parallel functional units with utilization of Uf
.....
PROCESSOR P
Execution of a single thread with utilization Up
F parallel functional units with utilization of Uf
CACHE
CACHE
MEMORY 1
Cycle time Dmem
MEMORY P
Cycle time Dmem
INTERCONNECTION NETWORK
Read/write requests/replies to/from remote memories
Latency Cnet clock cycles

Fig. 2. Block diagram of the NUMA approach. (P=number of processors, F=number of functional units, C_{net}=latency of the intercommunication network, D_{mem}=cycle time of the memory, U_p=utilization of processors, U_f=utilization of functional units.)

Cache coherent non-uniform memory access (CC-NUMA) is a variant of NUMA in which multiple processors with coherent caches and memory banks are connected together via a intercommunication network (memory is distributed, also remote memory is cached, coherence is managed by distributed directories, see Fig. 3) [Lenoski92]. Non-local memory accesses have higher (distance and traffic situation dependent) latency than local accesses. A typical CC-NUMA system consists of interconnected nodes, which have a processor, local caches and a local portion of main memory [Lenoski92, Agarwal95]. Introducing caching of remote memory leads to the issue of cache coherence. A cache line can have multiple copies distributed in all the caches in the system. Updating these cache lines on writes to make the write visible to all processors in the system is no trivial task. The cache coherence protocol specifies how the memory is maintained coherent. CC-NUMA cache coherence protocols are typically based on directories to keep track of the cache line copies in the system. The local cache controller in each node must consult a directory to fetch the most recent copy of the cache line on cache misses. Cache coherence protocols are known to be complicated and require advanced cache controllers.

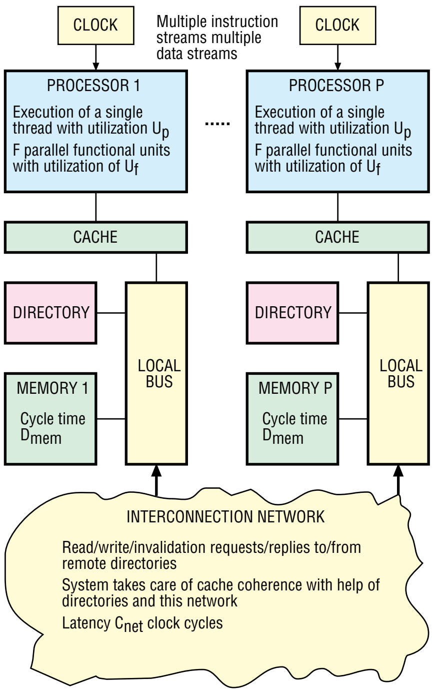

CLOCK
Multiple instruction streams multiple data streams
CLOCK
PROCESSOR 1
Execution of a single thread with utilization U_p
F parallel functional units with utilization of U_f
.....
PROCESSOR P
Execution of a single thread with utilization U_p
F parallel functional units with utilization of U_f
CACHE
CACHE
DIRECTORY
DIRECTORY
LOCAL BUS
LOCAL BUS
MEMORY 1
Cycle time D_{mem}
MEMORY P
Cycle time D_{mem}
INTERCONNECTION NETWORK
Read/write/invalidation requests/replies to/from remote directories
System takes care of cache coherence with help of directories and this network
Latency C_{net} clock cycles

Fig. 3. Block diagram of the CC-NUMA variant. (P=number of processors, F=number of functional units, C_{net}=latency of the intercommunication network, D_{mem}=cycle time of the memory, U_p=utilization of processors, U_f=utilization of functional units.)

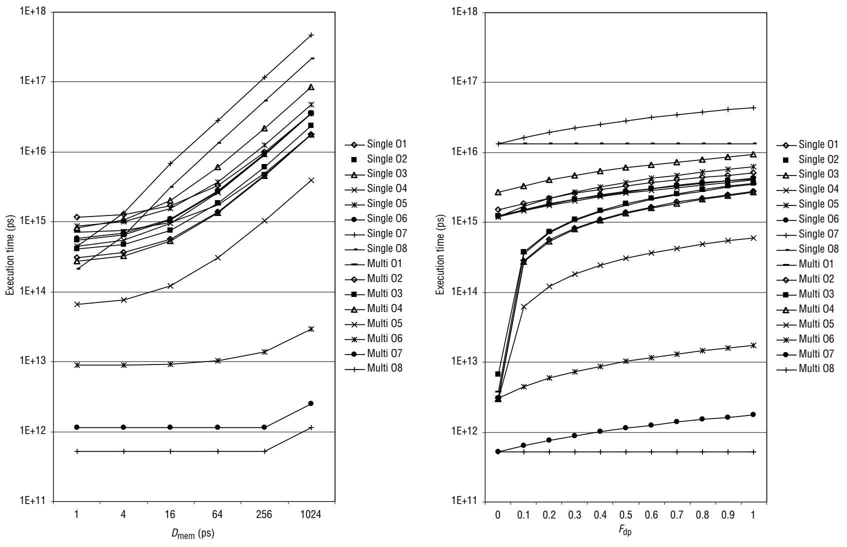
In our earlier performance evaluations and in an on-going efforts we model(ed) the performance and in some cases also silicon area and power consumption of different processor organizations in single and multiprocessor constellations, different approaches for TLP computing executing parametric benchmark application. In the following we summarize some findings/early results that reveal interesting things about the performance of the SMP and NUMA approaches with respect to other existing approaches and our proposals for efficient TLP-aware architectures:

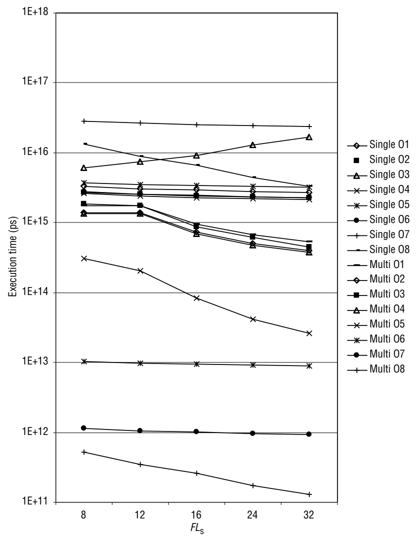
• In [Forsell02] we modeled analytically the performance of eight processor organizations in both single processor-memory and high-bandwidth multiprocessor constellations with different pipeline, ILP exploitation, and threading schemes (see Fig.4). With the used parameters, these models suggest that memory delays in multiprocessor systems are hidden best with multithreading and that the more dependencies the weaker performance except for the in-order multithreaded inter-thread pipelined chained FUs organization.

• In [Forsell11a] we measured the performance of multicore machines with different approaches to TLP, on-chip intercommunication topology, hashing schemes, and memory module organizations (see Fig. 5).

• In [Forsell11c] and related deliverables we are studying the performance of SMP, NUMA and ESM machines and code size with respect to different versions of parallel algorithms (see Fig. 6 for the very first results). The ESM approach makes use of the techniques explained in Sections 3-7 [Forsell06].

• In an on-going effort we are modeling analytically the performance of seven TLP approaches with a parametric workload (see Fig. 7 for early results). The ESM and CESM are approaches making use of the techniques explained in Sections 3-7 [Forsell06, Forsell09, Forsell10].





O1=in-order superscalar non-pipelined
O2=out-of-order superscalar
O3=superpipelined
O4=two-stage pipelined
O5=in-order superscalar chained FUs
O6=out-of-order multithreaded VLIW
O7=in-order multithreaded inter-thread pipelined
O8=in-order multithreaded inter-thread pipelined chained FUs.

Fig. 4. Execution time as a function of the memory system delay D_{mem}, the fraction of dependent parallel code F_{dp}, and the number of pipeline segments FL_{s} in single and multiprocessor constellations of eight processor organizations. For more details, see [Forsell02].)

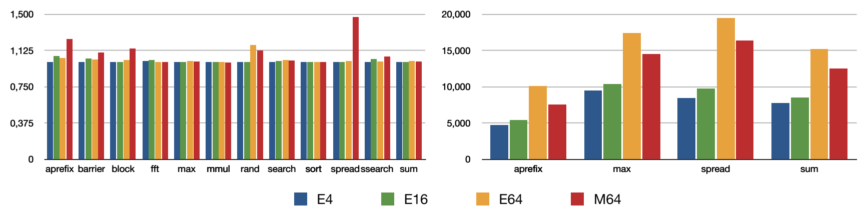


Fig. 5. Shared memory emulation overheads for certain simple functionalities optimized for MCRCW ESM and NUMA approaches. E4, E16 and E64 are 4, 16, and 64-core systems with a sparse mesh-based interconnect, M64 is a 64-core system with a multimesh interconnect, (For more details, see [Forsell11a].)

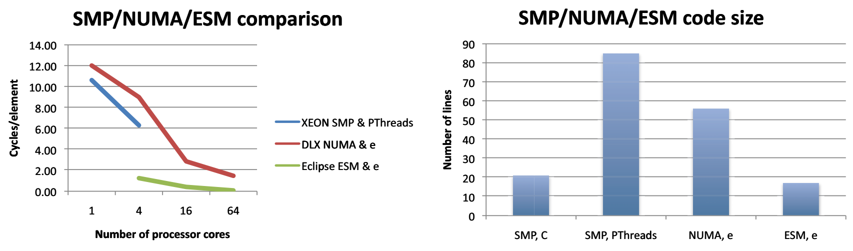


Fig. 6. Execution time and number of source code line of the aprefix benchmark in Xeon SMP using PThreads, DLX NUMA using the e-language and Eclipse ESM using the e-language. (For more details, see [Forsell11c].)

Since these studies are pointing to huge performance differences between the current SMP and NUMA approaches and new ESM and CESM approaches, we will take a closer look at computing and parallelism and architectural techniques needed to implement it efficiently. Generally speaking, the essence of parallel computing is to divide the computational problem at hand to subproblems that can be solved in parallel and to somehow solve the original problems with a help of these. This requires granting sufficient bandwidth for interprocessor communication, providing fast synchronization mechanism between subproblem executions, and making use of computational patterns that are specific to parallel computing only. These are discussed in the following four sections.

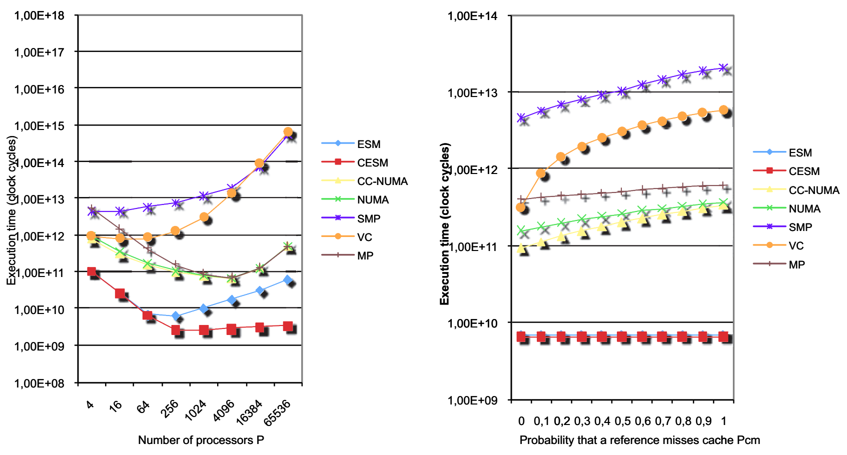


Fig. 7. Early estimations of the execution time of a parametric benchmark program as a function of number of processors (10% of code is sequential, 45% independent parallel, 45% dependent parallel, 20% of code is non-vectorizable, 10% accesses miss the cache, there are 8192 software threads) and the cache miss rate (10% of code is sequential, 45% independent parallel, 20% of code is non-vectorizable, there are 8192 software threads and 64 processors). (ESM=emulated shared memory, CESM=configurable emulated shared memory, CC-NUMA=cache coherent non-uniform memory access, NUMA=non-uniform memory access, VC=vector computing, MP=message passing)

3. Hiding the latency

As explained in Section 2, shared memory can be unified or distributed. A simple lay-out analysis reveals that unified shared memory will lead to delays proportional to the square root of the number of processor cores slowing down the memory system clock cycle and/or increasing the latency with respect to that of distributed shared memory interconnected with a mesh-like network (see Fig. 8). Another problems is the implementation of multiported memory. According to our studies the wiring of P-ported memory takes P^2 times more space than that of a single port memory of the same size. Thus, in the following we focus on distributed shared memory implementation.

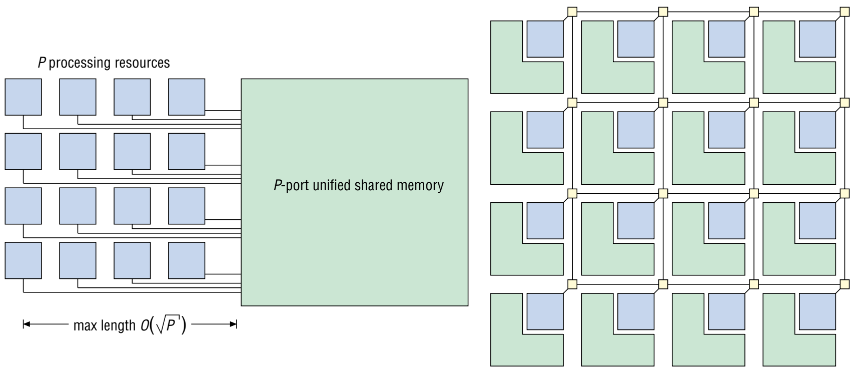


Fig. 8. A unified shared memory and distributed shared memory organization.

Accessing the memory on a distributed shared memory system takes time for communication and memory access on the target module. There are two principal means for hiding the latency of the memory system—caches and exploitation of parallel slackness.

Caches are small and fast associatively accessed memories that are widely used to balance the speed difference between processors and the main memory by keeping the potentially most frequently used data in the cache from which it can be quickly retrieved [Kilburn62]. In a case of a memory access, data is first searched from or stored to the cache. If it is found, a cache hit occurs and data can be delivered fast back to the processor. If data is not found, a cache miss occurs and an access to the slower main memory is performed. After the access is completed data is delivered to both the processor and stored into the cache so that further references to the same data can be performed faster. The main problem with caching applied to distributed shared memory is that if caches are placed in the front of the network, there is a need for cache coherency maintenance dropping the performance of the system with communication intensive algorithms [Lenoski28, Forsell10], or if caches placed in the behind of the network next to memory modules, they are not able to hide the latency of the intercommunication network.

The idea of latency hiding with *exploitation of parallel slackness* (or *high-throughput computing*) is that if multiple (denoted with T_p) threads are assigned to a single physical processor it is possible to execute other threads while a thread is accessing memory or committing communication (see Fig. 9). Special multithreaded processors and a pipelined memory system are needed to implement this efficiently [Ranade91, Leppänen96, Sun05]. Another advantage of high-throughput computing is that it slows down the execution of individual threads which helps to eliminate the pipeline hazards that slow down execution in single threaded architectures except in the minimal pipeline architecture in which there are not hazards [Forsell96, Forsell02] (see Fig. 10).

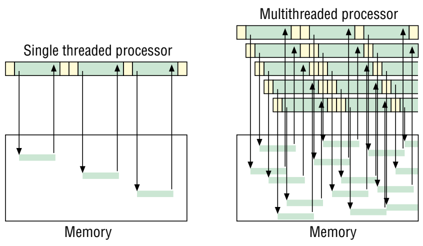


Fig. 9. Single and multithreaded execution and latency hiding with parallel slackness in the latter.

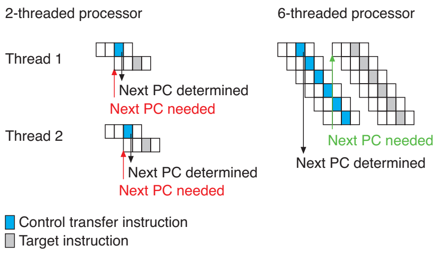


Fig. 10. Increasing the number of threads per processor from 2 to 6 eliminates control hazards in a 4-stage pipeline.

The effect of using high-throughput computing instead of caching is shown in Fig. 4 for different processor organizations and in Fig. 7 for TLP different approaches.

4. Synchronization

Synchronization is one of the most expensive operations in current parallel architectures. Even in a small multicore machine using the SMP or NUMA paradigm, a barrier synchronization can easily take hundreds of clock cycles. This rules fine-grained parallel algorithms useless for these architectures. There exists, however, very efficient synchronization techniques, e.g. the *synchronization wave*, for high-throughput architectures dropping the cost of implicit synchronization down to $1/T_p$ [Leppänen96] (see Fig.11).

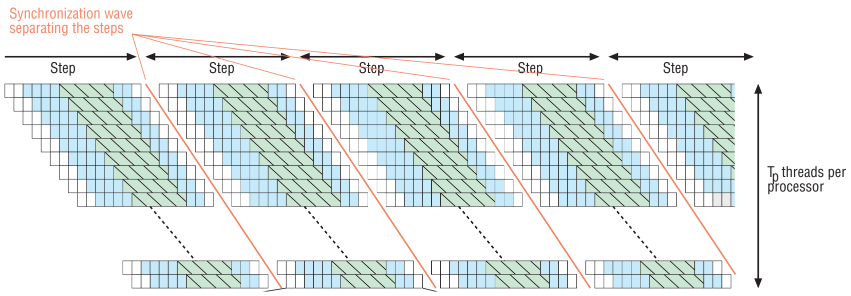


Fig. 11. Synchronization waves separating execution steps in high-throughput computing.

In it special synchronization messages are sent by the processors to the memory modules and vice versa. The idea is that when a processor has sent all its messages belonging to a single step on their way, it sends a synchronization message. Synchronization messages from various sources push on the actual messages, and spread to all possible paths, where the actual messages could go. When a switch receives a synchronization message from one of its inputs, it waits, until it has received a synchronization message from all of its inputs, then it forwards the synchronization wave to all of its outputs. The synchronization wave may not bypass any actual messages and vice versa. When a synchronization wave sweeps over a network, all switches, modules and processors receive exactly one synchronization message via each input link and send exactly one via each output link. This kind of a cheap implicit synchronization makes it possible to emulate the easy-to-program PRAM abstraction [Fortune78] efficiently and therefore these architectures are called emulated shared memory machine (ESM).

5. Exploitation of low-level parallelism

The amount of available ILP is often severely limited, especially if independent operations are not seek beyond the basic block boundaries [Jouppi89, Smith89, Forsell02]. Extracting larger amounts of ILP from a single threaded general purpose program is possible, but it requires very complex compilation algorithms like trace scheduling [Fisher81], superblock scheduling [Hank93] or percolation scheduling [Nicolau85, Moon93] seeking independent instructions across the boundaries of basic blocks. Even these advanced techniques fail, if instructions of the code are excessively dependent on each other. In high-throughput computing architectures it is possible to exploit ILP-style low level parallelism that often exists between the threads using the chained organization of functional units rather than traditional parallel organization [Forsell97] (see Fig. 12). The idea is to divide an instruction to a fixed number of subinstructions, which are executed in the corresponding functional units in a chain-like manner sequentially. This kind of technique, *chaining*, allows a unit to use the results of the preceding units so that a portion of strictly sequential subinstruction can be executed during a single instruction [Forsell97]. This is not possible with models based on parallel organization of functional units. We call the obtained ability to execute multiple subinstructions during a step as *virtual ILP*.

We have evaluated the effect of chaining both with analytic modeling [Forsell02] (see Fig. 4) and simulations [Forsell03] (see Fig. 13). Note that synchronous high-throughput computing works poorly with dynamic superscalar execution since the scheduler can easily change the ordering and timing of memory accesses so that the dependencies are violated and erroneous result is obtained.

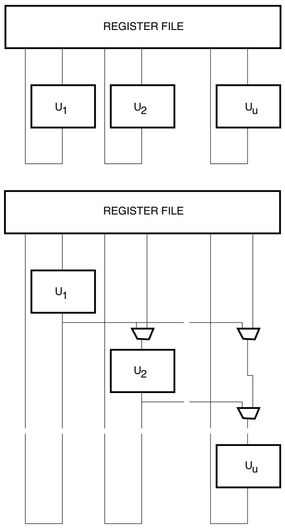


Fig. 12. Parallel and chained organization of functional units.

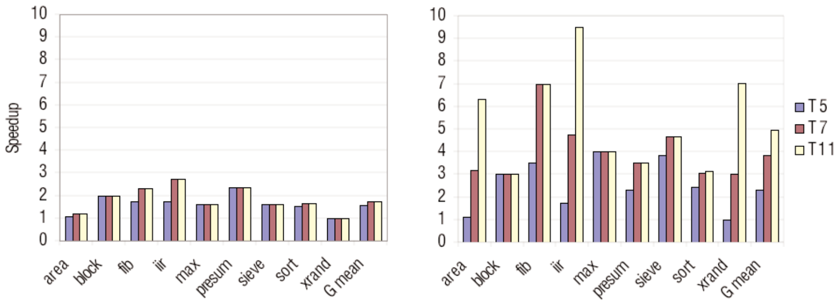


Fig. 13. Speedup of achieved by using parallel and chained organization of functional units with 5, 7 and 11 functional units per processor with respect to standard pipelined processor organization.

6. Concurrent memory access

Memory access is said to be *concurrent* if at least two processors are accessing the same location during the same step. Naturally, this kind of concurrence is meaningful only in the case of synchronous TLP execution, e.g. in the synchronous high-throughput computing making use of parallel slackness in latency hiding and synchronization wave as described above but not in SMP and NUMA. According to the theory of parallel algorithms [Jaja92], concurrent memory accesses can be used to speed up execution by a logarithmic factor for a class of parallel algorithms. There are efficient techniques for concurrent memory access even if we limit the number of ports per memory module to one—namely combining and step caches [Keller01, Forsell05].

Combining is a technique in which memory references targeted to the same location are combined during routing so that traffic to the final destination is reduced [Ranade91]. Unfortunately, there is a need to sort the references made during a step of (multithreaded) execution on a processor prior sending them to the network, a need for machinery in the routers comparing references on their way to destinations, and a need to store combined read messages so that their common reply can be split back to actual replies targeted to processing resources [Keller01]. All this makes implementation of combining expensive and tricky.

Step caches [Forsell05] are special caches that operate like normal caches but data stored to a step cache remains valid only until the end of ongoing step of multithreaded execution during which the references are independent by the definition of parallel computing so that there is no need to maintain any kind of cache coherence. In a system with P T_p-threaded processor cores, step caches help concurrent memory accesses by reducing the number of references per a location from PT_p to P, which allows the memory system to work without a noticeable performance penalty if $T_p \geq P$ even if all the threads are referring to the same location in parallel (see Fig. 14). Compared to combining, step caches operate faster due to elimination of the sorting phase but require more parallelism and operate more arbitrarily, e.g. not preserving the ordering of memory references and thus making in more difficult to implement e.g. the Priority variant of the concurrent memory access aware PRAM in which the thread with the lowest id wins in the case of a concurrent write. Fig. 15 shows our simulation results [Forsell08a] for a step cached ESM as the speedups achieved with a help of concurrent memory access in the case of 4, 16 and 64 processor cores with respect to 4-core exclusive access.

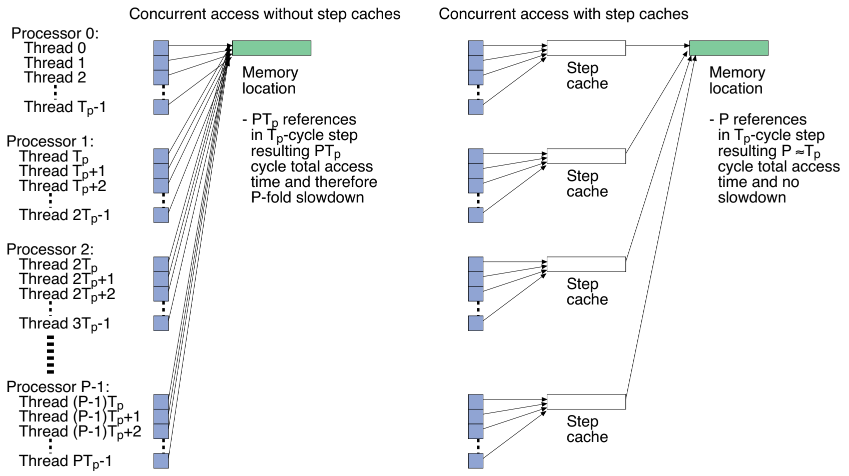


Fig. 14. Concurrent memory access in high-throughput computing approach with and without step caches.

	E4	C4	E16	C16	E64	C64
aprefix	1.00	1.41	3.18	4.83	9.49	14.63
max	1.00	1.38	3.16	4.74	12.75	16.31
search	1.00	12.09	3.26	48.12	12.37	189.07
spread	1.00	72.00	3.18	287.43	11.16	1136.46
sum	1.00	1.41	3.18	4.83	11.13	14.63

Fig. 15. Speedup provided by concurrent memory access with respect to E4. (Ex=exclusive memory access with x processor cores, Cx=concurrent memory access with x processor cores)

7. Multioperations

According to the theory of parallel algorithms the logarithmic speedup provided by concurrent memory access (CRCW) can be extended to a wider set of algorithms if efficient implementation of multi(prefix)operations operations is provided for synchronous TLP machines [Jaja92]. A *multioperation* is a special operation involving participation of multiple threads and a single memory location, e.g. summing up to *P* data values to a memory location in parallel. *Multiprefix operations* are like multioperations, but each processor receives a cumulative intermediate result of the operation. Both multioperations and multiprefix operations can be implemented in constant time with a combining networks technique [Ranade91] or with step caches/scratchpad and active memory units [Forsell06, Forsell11b]. The difference of combining and step cache/scratchpad-based techniques is that while the latter takes potentially less silicon area and performs faster in cases not requiring full concurrency, it is not able to provide unbounded number of concurrent ordered multiprefixes like combining. Fig. 16 illustrates an implementation of ordered multiprefix with a help of step caches, scratchpads and active memory unit ordering buffers. Fig. 17 shows our simulation results [Forsell08a] as the speedups achieved with a help of multioperations in the case of 4, 16 and 64 processor cores with respect to 4-core exclusive access without multioperations.

8. Conclusions

Based on our recent and on-going analysis of TLP execution techniques, we have tried to answer the question why the approaches used in current multicore machines, making use of performance enhancement techniques primarily developed for sequential computers, are not efficient in TLP execution. For that we have introduced alternative/new TLP-aware techniques that give much higher performance for many general purpose TLP computing patterns. The performance advantage comes from scalable latency hiding, radically faster synchronization, more efficient exploitation of low-level parallelism, and special synchronous TLP execution aware techniques like concurrent memory access and multi(prefix)operations. Thus, preliminary speaking we can say that high-throughput computing style interleaved multithreading could replace caching as a primary latency hiding mechanism for shared memory access but caching would remain important at the level of memory hierarchy related

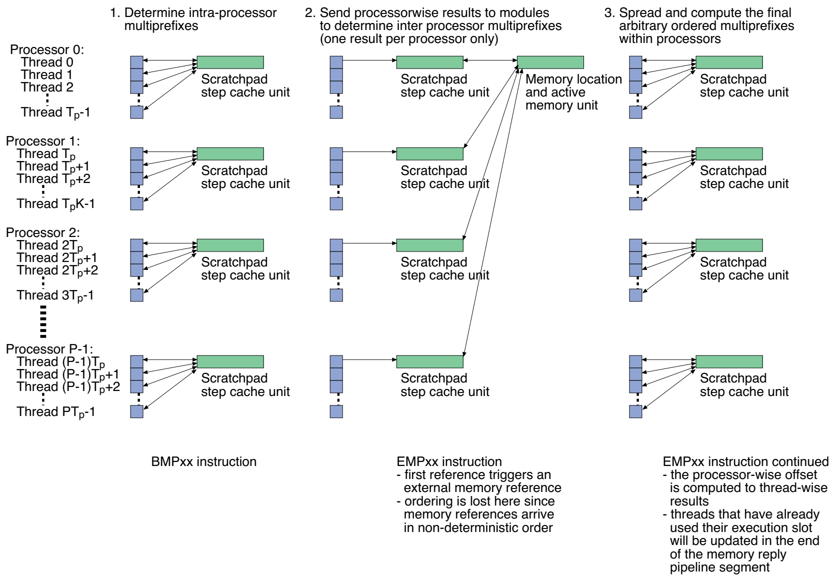

1. Determine intra-processor multiprefixes
2. Send processorwise results to modules to determine inter processor multiprefixes (one result per processor only)
3. Spread and compute the final arbitrary ordered multiprefixes within processors
Processor 0:
Thread 0
Thread 1
Thread 2
Thread T_p-1
Processor 1:
Thread T_p
Thread T_p+1
Thread T_p+2
Thread T_pK-1
Processor 2:
Thread 2T_p
Thread 2T_p+1
Thread 2T_p+2
Thread 3T_p-1
Processor P-1:
Thread (P-1)T_p
Thread (P-1)T_p+1
Thread (P-1)T_p+2
Thread PT_p-1
Scratchpad step cache unit
Memory location and active memory unit
BMPxx instruction
EMPxx instruction
- first reference triggers an external memory reference
- ordering is lost here since memory references arrive in non-deterministic order
EMPxx instruction continued
- the processor-wise offset is computed to thread-wise results
- threads that have already used their execution slot will be updated in the end of the memory reply pipeline segment

Fig. 16. Ordered multiprefix in high-throughput computing approach with and without step caches.

	E4	M4	E16	M16	E64	M64
aprefix	1.00	11.99	3.18	47.27	9.49	184.07
max	1.00	9.85	3.16	39.05	12.75	151.90
search	1.00	12.08	3.26	44.83	12.37	188.90
spread	1.00	71.93	3.18	287.17	11.16	1135.45
sum	1.00	14.39	3.18	56.02	11.13	221.74

Fig. 17. Speedup provided by multioperations. (Ex=exclusive memory access with x processor cores, Mx=multioperation and concurrent memory access aided execution with x processor cores)

accessing slower/secondary memories from distributed memory modules, and chained static virtual superscalar execution could replace dynamic superscalar execution as the most efficient techniques for low-level parallelism exploitation. Techniques not familiar from sequential nor current TLP machines—concurrent memory access and multioperations—could provide further boost enabling advanced parallel patterns. Likewise, techniques that would become less important include some standard pipeline hazard prevention mechanisms and speculations. Since this kind of elimination of performance bottlenecks makes it possible to use fine-grained TLP programming patterns efficiently, this will also simplify programming considerably. All this may considerably change the theory and practice of parallel computing in the near future.

Even though this paper is based on partially preliminary models and on-going work we believe that they point already relatively well at the right direction. Detailed and verified analytic performance models of TLP approaches and thorough discussion on implications of these will be published in the later phases of our research.

References

[Agarwal95] A. Agarwal et al., The MIT Alewife machine: Architecture and Performance, Proceedings of the 22nd annual international symposium on computer architecture (ISCA), 1995, pp. 2-13.

[Bloch59] E. Bloch, The engineering design of the Stretch computer, Proceedings of the Fall Joint Computer Conference, 1959, 48-59.

[Brooks00] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta and P. Cook, Power-Aware Microarchitecture: Design and Modeling Challenges for Next Generation Microprocessors, IEEE Micro 20, 6 (November-December 2000), 26-32.

[Bucholz62] W. Bucholz, Planning a Computer System: Project Stretch, McGraw-Hill, New York, 1962.

[Culler99] D. Culler and J. Singh, Parallel Computer Architecture—A Hardware/ Software Approach, Morgan Kaufmann Publishers Inc., San Fransisco, 1999.

[Fisher81] J. Fisher, Trace Scheduling: A Technique for Global Microcode Compaction, IEEE Transactions on Computers C-30, 7 (July 1981), 478-490.

[Forsell96] M. Forsell, Minimal Pipeline Architecture-an Alternative to Superscalar Architecture, Microprocessors and Microsystems 20, 5 (1996), 277-284.

[Forsell97] M. Forsell, MTAC—A Multithreaded VLIW Architecture for PRAM Simulation, Journal of Universal Computer Science 3, 9 (1997), 1037- 1055.

[Forsell02] M. Forsell, Architectural differences of efficient sequential and parallel computers, Journal of Systems Architecture 47, 13 (July 2002), 1017-1041.

[Forsell05] M. Forsell, Step Caches—a Novel Approach to Concurrent Memory Access on Shared Memory MP-SOCs, In the Proceedings of the 23th IEEE NORCHIP Conference, November 21-22, 2005, Oulu, Finland, 74-77.

[Forsell06] M. Forsell, Realizing Multioperations for Step Cached MP-SOCs, In the Proceedings of the International Symposium on System-on-Chip 2006 (SOC'06), November 14-16, 2006, Tampere, Finland, 77-82.

[Forsell08a] M. Forsell, On the performance and cost of some PRAM models on CMP hardware, In the Proceedings of the 10th Workshop on Advances in Parallel and Distributed Computational Models (in conjunction with the 22th IEEE International Parallel and Distributed Processing Symposium, IPDPS´08), April 14, 2008, Miami, USA.

[Forsell08b] M. Forsell, V. Leppänen and M. Penttonen, Rinnakkaistietokoneen uusi tuleminen, Tietojenkäsittelytiede 28, (Joulukuu2008), 55-65.

[Forsell09] M. Forsell, Configurable Emulated Shared Memory Architecture for general purpose MP-SOCs and NOC regions, In the Proceedings of the 3rd ACM/IEEE International Symposium on Networks-on-Chip, May 10-13, 2009, San Diego, USA, 163-172.

[Forsell10] M. Forsell, A PRAM-NUMA model of computation for addressing low-TLP workloads, in The Proceedings of the 12th Workshop on Advances in Parallel and Distributed Computational Models (in conjunction with the 24th IEEE International Parallel and Distributed Processing Symposium, IPDPS'10), April 19, 2010, Atlanta, USA, 1–8.

[Forsell11a] M. Forsell, Performance comparison of some shared memory organizations for 2D mesh-like NOCs, Microprocessors and Microsystems 35, 2 (March 2011), 274-284.

Ten Years of Bluetooth Security Attacks: Lessons Learned

Keijo Haataja, Konstantin Hyppönen, Pekka Toivanen
School of Computing
University of Eastern Finland
P.O. Box 1627, 70211 Kuopio, Finland
Keijo.Haataja@uef.fi, Konstantin.Hypponen@uef.fi,
Pekka.Toivanen@uef.fi

Abstract. In this paper, a literature review based comparative analysis of Bluetooth security attacks over the past ten years (2001-2011) is provided. In addition, a new practical countermeasure against Man-In-The-Middle attacks on Bluetooth Secure Simple Pairing is proposed. Moreover, a novel attack that works against all existing Bluetooth versions is proposed. Furthermore, some new ideas that will be used in our future research work are presented.

1 INTRODUCTION

Bluetooth[1] is a technology for short range wireless data and real-time two-way audio/video transfer providing data rates up to 24 Mb/s. Bluetooth operates at 2.4 GHz frequency in the free Industrial, Scientific, and Medical (ISM) band. Bluetooth devices that communicate with each other form a *piconet*. The device that initiates a connection is the piconet *master* and all other devices within that piconet are *slaves*.

Already in 2006, the one billionth Bluetooth device was shipped [1]. Less than five years later in 2011, the four billionth Bluetooth device was shipped[2], and the volume is expected to increase rapidly in the near future. According to In-Stat, the eight billionth Bluetooth device is expected to be shipped by the end of 2013[3]. Therefore, it is very important to keep Bluetooth security issues up-to-

[1]https://www.bluetooth.org/Technical/Specifications/adopted.htm

date.

Our results: In this paper, we provide a literature review based comparative analysis of Bluetooth security attacks over the past ten years (2001-2011). In addition, we propose a new practical countermeasure against Man-In-The-Middle (MITM) attacks on Bluetooth Secure Simple Pairing (SSP). Moreover, we propose a new practical attack that works against all existing Bluetooth versions. Furthermore, we present some new ideas that will be used in our future research work.

The rest of the paper is organized as follows. Section 2 gives an overview of Bluetooth security. A literature review based comparative analysis of Bluetooth security attacks over the past ten years (2001-2011) is provided in Section 3: the attacks are designed against Bluetooth versions up to 2.0+EDR (Enhanced Data Rate), but some of them work also against all existing Bluetooth versions up to 4.0. Since MITM attacks are also possible and dangerous against the latest SSP-enabled Bluetooth versions (i.e. Bluetooth versions 2.1+EDR – 4.0), MITM attacks on Bluetooth are explained in a separate attack section (Section 4). The section also provides a comparative analysis of the all existing Bluetooth MITM attacks over the past ten years (2001-2011). Section 5 proposes a new practical countermeasure against MITM attacks on SSP. A new practical attack that works against all existing Bluetooth versions is proposed in Section 6. Finally, Section 7 concludes the paper and sketches future work.

2 OVERVIEW OF BLUETOOTH SECURITY

The basic Bluetooth security configuration is done by the user who decides how a Bluetooth device will implement its connectability and discoverability options. The different combinations of connectability and discoverability capabilities can be divided into three categories, or *security levels:*[1] [1]

[2]http://www.bluetooth.com/Pages/Press-Releases-Detail.aspx?ItemID=126
[3]http://www.instat.com/press.asp?Sku=IN1104968MI&ID=3238

1. *Silent:* The device will never accept any connections. It simply monitors Bluetooth traffic.

2. *Private:* The device cannot be discovered, i.e. it is a so-called *non-discoverable device*. Connections will be accepted only if the *Bluetooth Device Address (BD_ADDR)* is known to the prospective master. A 48-bit BD_ADDR is normally unique and refers globally to only one individual Bluetooth device.

3. *Public:* The device can be both discovered and connected to. It is therefore called a *discoverable device*.

Bluetooth security is based on building a chain of events, none of which should provide meaningful information to an eavesdropper. All events must occur in a specific sequence for security to be set up successfully. In order for two Bluetooth devices to start communicating, a procedure called *pairing* must be performed. As a result of pairing, two devices form a trusted pair and establish a link key which is used later on for creating a data encryption key for each session.[1] [1]

In Bluetooth versions up to 2.0+EDR, pairing is based exclusively on the fact that both devices share the same *Personal Identification Number (PIN)*, or *passkey*, that is used for generating several 128-bit keys. When the user enters the same passkey in both devices, the devices generate the same shared secret which is used for authentication and encryption of traffic exchanged by them. The PIN is the only source of entropy for the shared secret in Bluetooth versions up to 2.0+EDR. As the PINs often contain only four decimal digits, the strength of the resulting keys is not enough for protection against passive eavesdropping on communication. Even with longer 16-character alphanumeric PINs, full protection against active eavesdropping cannot be achieved: it has been shown that MITM attacks on Bluetooth communications (versions up to 2.0+EDR) can be performed [1–4].[1] [1]

Bluetooth versions 2.1+EDR, 3.0+HS (High-Speed), and 4.0 add a new specification for the pairing procedure, namely SSP[1]. Its main goal is to improve the security of pairing by providing protection against passive eavesdropping and MITM attacks. Instead of using (often short) passkeys as the only source of entropy for building the link keys, SSP employs Elliptic Curve Diffie-Hellman (ECDH) public-key cryptography. To construct the link key, devices use public-private key pairs, a number of nonces, and Bluetooth addresses of the devices. Passive eavesdropping is effectively thwarted by SSP, as running an exhaustive search on a private key with approximately 95 bits of entropy is currently considered to be infeasible in short time.[1] [1]

In order to provide protection against MITM attacks, SSP either uses an OOB channel (e.g., Near Field Communication, NFC), or asks for the user's help: for example, when both devices have displays and keyboards, the user is asked to compare two six-digit numbers. Such a comparison can be also thought as an OOB channel which is not controlled by the MITM. If the values used in the pairing process have been tampered with by the MITM, the six-digit integrity checksums will differ with the probability of 0.999999.[1] [1]

SSP uses four *association models*. In addition to the two association models mentioned previously, *OOB* and *Numeric Comparison*, models named *Passkey Entry* and *Just Works* are defined. The Passkey Entry association model is used in the cases when one device has input capability, but no screen that can display six digits. A six-digit checksum is shown to the user on the device that has output capability, and the user is asked to enter it on the device with input capability. The Passkey Entry association model is also used if both devices have input, but no output capabilities. In this case the user chooses a 6-digit checksum and enters it in both devices. Finally, if at least one of the devices has neither input nor output capability, and an OOB cannot be used, the Just Works association model is used. In this model the user is not asked to perform any operations on numbers: instead, the device may simply ask the user to accept the connection.[1] [1]

SSP is comprised of six phases:[1] [1]

1. *Capabilities exchange:* The devices that have never met before or want to perform re-pairing for some reason, first exchange their Input/Output (IO) capabilities to determine the proper association model to be used.

2. *Public key exchange:* The devices generate their public-private key pairs and send the public keys to each other. They also compute the Diffie-Hellman key.

3. *Authentication stage 1:* The protocol that is run at this stage depends on the association model. One of the goals of this stage is to ensure that there is no MITM in the communication between the devices. This is achieved by using a series of nonces, commitments to the nonces, and a final check of integrity checksums performed either through the OOB channel or with the help of user.

4. *Authentication stage 2:* The devices complete the exchange of values (public keys and nonces) and verify the integrity of them.

5. *Link key calculation:* The parties compute the link key using their Bluetooth addresses, the previously exchanged values, and the Diffie-Hellman key constructed in phase 2.

6. *LMP authentication and encryption:* Encryption keys are generated in this phase, which is the same as the final steps of pairing in Bluetooth versions up to 2.0+EDR.

Even though SSP improves the security of Bluetooth pairing, it has been shown that MITM attacks against Bluetooth 2.1+EDR, 3.0+HS, and 4.0 devices are also possible by forcing victim devices to use the Just Works association model [1,5–9]. Thus, the security of SSP should be further improved.

3 COMPARATIVE ANALYSIS OF BLUETOOTH SECURITY ATTACKS

Security threats in distributed networks (such as Bluetooth) can be divided into three categories: disclosure threat, integrity threat, and Denial-of-Service (DoS) threat. *Disclosure threat* means that information can leak from the target system to an eavesdropper that is not authorized to access the information. *Integrity threat* concerns the deliberate alteration of information in an attempt to mislead the recipient. *DoS threat* involves blocking access to a service, making it either unavailable or severely limiting its availability to an authorized user. Disclosure and integrity attacks typically compromise some sensitive information and therefore can be very dangerous, while DoS attacks typically only annoy Bluetooth network users and are considered to be less dangerous. [1,10]

Sections 3.1, 3.2, and 3.3 explain some typical disclosure, integrity, and DoS threats, respectively. Some typical threats which cannot be classified as only one single threat (so-called *multithreats*) are explained in Section 3.4.

3.1 Disclosure Threats

BlueSnarfing attack [4,5] (also referred to as *BlueStumbling attack*) means that an attacker connects to the target device without alerting its owner and steals some sensitive information, such as entire phonebook, calendar notes, and text messages. At least three BlueSnarfing applications exist: Adam Laurie's *BlueSnarf* [5], Ollie Whitehouse's *RedSnarf* [6], and Bluediving Project's *Bluediving* [7]. The success of BlueSnarfing attack depends very much on the vendor's implementation of the Bluetooth protocol stack for the target device. Therefore, the attack works only if the protocol stack of the target device is poorly implemented, i.e. there are serious flaws in the authentication and data transfer mechanisms of some Bluetooth devices. A list of the devices known to be vulnerable to BlueSnarfing attack without firmware/software update can be found in [5]. [1]

An *Off-Line PIN Recovery attack* [6] [2] (also referred to as *Off-Line PIN Crunching attack*) is based on intercepting the traffic of the initial pairing process and after that trying to calculate the correct SRES (Signed Response; i.e. authentication result) value by guessing different PIN values until the calculated SRES equals to the intercepted SRES. It is worth noting that SRES is only 32 bits long. Therefore, a SRES match does not necessarily guarantee that an attacker has discovered the correct PIN code, but the chances are quite high especially if the PIN code is short. The attack is dangerous only if the PIN code is short and it has no case-sensitive alphanumerical characters (and perhaps some other characters as well). The requirement to witness the initial pairing process between the victim devices is not a big problem for the attacker, because it can be arranged in many different ways: for example, by disrupting the Physical Layer (PHY) in such a way that a frustrated user thinks something is wrong and deletes previously stored link keys, by using three methods proposed in [11] which can force two target devices to repeat the initial pairing process, or by sending a Bluetooth device anonymously to the target person as a prize in some competition. After that the user initiates a new pairing process and the attacker can intercept all the required inputs for an Off-Line PIN Recovery attack.[6] [1,2,10,12]

An *Enhanced implementation of Off-Line PIN Recovery attack* [11] is an average of 30 % faster than the original Off-Line PIN Recovery attack[6] [2]. It is based on the optimization of Secure And Fast Encryption Routine + (SAFER+) [13] using the algebraic manipulation of the SAFER+ round. Moreover, three methods which can *force two target devices to repeat the initial pairing process* are also proposed in [11]: the methods are based on the fact that Bluetooth

[4]http://trifinite.org/Downloads/BlueSnarf_CeBIT2004.pdf

[5]http://madrock.net/2008/03/serious-flaws-in-bluetooth-security-lead-to-disclosure-of-personal-data

[6]http://cansecwest.com/csw04archive.html (see the report of Ollie Whitehouse)

[7]http://bluediving.sourceforge.net

specifications[1] allow Bluetooth devices to forget a link key and thus an attacker can abuse this possibility by forcing victim devices to forget their current link key. However, the Bluetooth user is required to enter a PIN code again during the new pairing process and therefore a suspicious user may realize that her device is under attack. [1,11,13]

An *Off-Line Encryption Key Recovery attack* [6] extends an Off-Line PIN Recovery attack[6] [2,11]. When the PIN code is discovered via an Off-Line PIN Recovery attack, the attacker can produce the required ACO (Authenticated Ciphering Offset; i.e. a 96-bit authentication result) and thus she can also recover the encryption key. Therefore, an Off-Line Encryption Key Recovery attack is dangerous only when an Off-Line PIN Recovery attack or its enhanced implementation has been completed successfully.[6] [1]

A *Brute-Force BD_ADDR Scanning attack* [6] [1] uses a brute-force method only on the last three bytes of a BD_ADDR, because the first three bytes are publicly known and can be set as fixed. *RedFang* [8] is a security analysis tool for finding non-discoverable Bluetooth devices by brute-forcing the last three bytes of BD_ADDR and doing a name inquiry[6,8]. We also designed, implemented, and tested our own tool to carry out this attack. Our *Brute-Force BD_ADDR Scanning Security Analysis Tool [1,14] is on average four times faster than RedFang*, because it runs on a special hardware, LeCroy's Bluetooth protocol analyzer[9], which can use Bluetooth radio much more efficiently than a normal PC with a Bluetooth Universal Serial Bus (USB) dongle. A detailed description of our security analysis tool can be found in [1,14]. [1]

Besides a Brute-Force BD_ADDR Scanning attack, *techniques for finding hidden Bluetooth devices in an average of one minute* have been proposed [15] and even implemented in a form of an open-source Bluetooth sniffer[10] that operates on a single channel. The Universal Software Radio Peripheral (USRP[11]) was used as a radio device to

[8]http://www.securiteam.com/tools/5JP0I1FAAE.html

[9]http://www.lecroy.com/protocolanalyzer/protocoloverview.aspx?seriesid=103&capid=103&mid=511

eavesdrop on Bluetooth packets. It is the hardware device associated with the GNU Radio Project[12], which develops an open-source framework for implementing software radio systems, i.e. systems in which radio devices are implemented in software. Due to the buffering and asynchronous nature of the GNU Radio framework and the hardware restrictions of the USRP, no working prototype of the Bluetooth sniffer that supports frequency hopping has been implemented yet. However, the current version of the Bluetooth sniffer is still capable of finding hidden (non-discoverable) Bluetooth devices in the range of vulnerability in an average of one minute.[10] [1,15]

A *BluePrinting attack* [13] is used to determine the manufacturer, device model, and firmware version of the target device. For example, an attacker can use Blueprinting to generate statistics about Bluetooth device manufacturers and models, and to find out whether there are devices in the range of vulnerability that have issues with Bluetooth security. *BluePrint* [14] is a tool for performing BluePrinting attack under Linux environments.[13,14] [1]

Our practical experiment, *Interception of Packets attack* [1,16], was conducted in order to demonstrate the importance of data encryption and to show how easy it is for an eavesdropper to intercept all packets exchanged via air. Our practical experiment clearly demonstrated that all information exchanged via air can be seen clearly on the screen of the attacker's laptop when encryption is not used. A detailed description of our practical experiment can be found in [1,16].

3.2 Integrity Threats

Reflection attacks [4] (also referred to as *Relay attacks*) are based on the impersonation of target devices. An attacker does not have to

[10]http://www.cs.ucl.ac.uk/staff/a.bittau/gr-bluetooth.tar.gz
[11]http://www.ettus.com/downloads/ettus_ds_usrp_v7.pdf
[12]http://gnuradio.org/redmine/projects/gnuradio/wiki
[13]http://trifinite.org/Downloads/Blueprinting.pdf
[14]http://trifinite.org/trifinite_stuff_blueprinting.html

know any secret information, because she only relays (reflects) the received information from one target device to another during the authentication, i.e. a Reflection attack in Bluetooth can be seen as a type of a MITM attack against authentication, but not encryption. Section 4.2 provides more information about Reflection attacks. [1, 4]

A very dangerous form of integrity threat takes place when an attacker uses a stronger RF signal in order to displace the active piconet device: the main principle for successfully completing an *Exploitation of a stronger RF signal attack* [10] is to send the target device's receiver an RF signal that is at least 11 dB stronger than the signal that the legitimate piconet device is sending, thus capturing the channel from the legitimate piconet device. [1,10]

A *Backdoor attack* [5] means that an attacker establishes a trusted relationship with the target device through authentication and ensures that this trusted relationship no longer appears to be in the target device's register of paired (authenticated) devices. When the backdoor is installed in the target device via a Backdoor attack, the attacker can continue the attack in many different ways: for example, trying to exploit the resources of the target device via a trusted relationship, trying to perform a BlueSnarfing attack (see Section 3.1), or trying to slip a virus or worm to the target device (see Section 3.4). A list of the devices known to be vulnerable to Backdoor attacks without a firmware/software update can be found in [5]. [1]

3.3 DoS Threats

DoS threats can be roughly divided into two parts: *attacks against the PHY* and *attacks against protocols above the PHY*. [1,10]

At the Physical Layer (PHY), an attacker can jam the piconet entirely or capture the channel from the legitimate piconet device. A jammer can perform *Disruption of the PHY attack* [10] relatively far away from the communicating devices (approximately up to 100 meters) by using a Bluetooth transmitter with a power amplifier and a directional antenna [1,10]. This type of attack can be very dangerous if the attacker is using a stronger RF signal to displace the existing legitimate piconet device (see Section 3.2) and then trying to steal some sensitive information from the target device. [1,10]

Attacks on higher levels of the Bluetooth protocol stack try to exploit some of the characteristics of higher level protocols in an attempt to occupy the attention of one or more devices of the piconet in such a way that they are unable to serve other legitimate devices within a reasonable time. [1,10]

A *BD_ADDR Duplication attack* [1,10] is based on the idea that an attacker places a bug in the range of susceptibility. The bug duplicates the BD_ADDR of the target device. When any Bluetooth device tries to make a connection with the target device, either the target device or both devices, i.e. the target device and the bug, will respond and jam each other. In this way, the attacker has denied access from the legitimate device. We designed, implemented, and tested our own Bluetooth security analysis tool, *BD_ADDR Duplication Security Analysis Tool* [1,17], which was successfully used to perform BD_ADDR Duplication attacks. A detailed description of our security analysis tool can be found in [1,17]. [1]

A *Synchronous Connection-Oriented (SCO) / Extended SCO (eSCO) attack* [1,10] is based on the fact that a realtime two-way voice reserves a great deal of a Bluetooth piconet's attention so that the legitimate piconet devices are not getting the service within a reasonable time. We designed, implemented, and tested our own Bluetooth security analysis tool, *SCO/eSCO Security Analysis Tool* [1,17], which was successfully used to perform SCO attacks. A detailed description of our security analysis tool can be found in [1,17]. [1]

A *Big Negative Acknowledgement (NAK) attack* [1,10] is based on the idea of putting the target device on an endless retransmission loop so that the legitimate piconet devices have considerably slowed throughput. In the attack, an attacker requests any information from the target device and every time the requested information is received, the attacker sends NAK, i.e. the transmission has failed. We designed, implemented, and tested our own Bluetooth security analysis tool, *Big NAK Security Analysis Tool* [1,17], which was successfully used to perform Big NAK attacks. A detailed description of our security analysis tool can be found in [1,17]. [1]

A *Logical Link Control and Adaptation Protocol (L2CAP) Guaranteed Service attack* [1, 10] is based on the idea that an attacker requests the highest possible data rate or the smallest possible latency from the target device so that all other connections are refused and all throughput is reserved for the attacker. However, the success of the attack is implementation-dependent, because all Bluetooth devices do not necessarily support the abovementioned optional L2CAP Quality-of-Service (QoS) features. [1]

A *Bluetooth Object Exchange Protocol (OBEX)*[15] *Message attack* [16] is based on Nokia 6310i[17] Bluetooth mobile phone's flaw (some other Bluetooth mobile phones may also be vulnerable) that allows an attacker to perform a remote DoS attack. The attack can be performed by sending invalid Bluetooth OBEX messages to the target device. As a result of this attack, the target device will loss its availability and may crash/reboot.[16] [18]

A *BlueSmacking attack* [18] is based on using the standard tools that are shipped with BlueZ protocol stack[19]. An attacker can use BlueSmacking attack to knock out some Bluetooth devices immediately: for example, many HP's iPAQ Personal Digital Assistants (PDAs) are vulnerable to BlueSmacking attack.[18] [18]

A *BlueSpamming attack* [20] is based on the idea that an attacker spams Bluetooth devices with arbitrary files if they support OBEX: *BlueSpam* [20] is a Palm OS application for performing BlueSpamming attacks.[20] [18]

A *Battery Exhaustion attack* [1,10] is based on the idea of occupying the target device in such a way that it also consumes rather quickly the battery of the target device. [1,10]

[15]http://www.irda.org
[16]http://www.osvdb.org/displayvuln.php?osvdb_id=3890
[17]http://nds1.nokia.com/phones/files/guides/6310i_usersguide_en.pdf
[18]http://trifinite.org/trifinite_stuff_bluesmack.html
[19]http://www.bluez.org
[20]http://www.mulliner.org/palm/bluespam.php

3.4 Multithreats

There are also many attacks which cannot be classified as only one single threat. For example, BlueBugging attacks, Blooovering attacks, HeloMoto attacks, On-Line PIN Cracking attacks, BTKeylogging attacks, and BTVoiceBugging attacks can be classified as *disclosure and integrity threats*.

A *BlueBugging attack* [21] [1] means that an attacker connects to the target device (typically a Bluetooth mobile phone) without alerting its owner, steals some sensitive information, such as an entire phonebook, calendar notes, and text messages, and has full access to the GSM (Global System for Mobile communications) AT command set. It means that the attacker can, in addition to stealing information, send text messages to premium numbers, initiate phone calls to premium numbers, write phonebook entries, connect to the Internet, set call forwards, try to slip a Bluetooth virus or worm to the target device, and many other things. A list of the devices known to be vulnerable to a BlueBugging attack without a firmware/software update can be found in [5]. Several public BlueBugging tools exist: for example, *btxml* [22], *Blooover* [23], and *Blooover II* [24]. Our practical experiment, *BlueBugging attack* [1, 18], demonstrated the dangerousness of such an attack by using the btxml[22]. A detailed description of our practical experiment can be found in [1, 18].[21] [1]

Blooover[23] and its successor Blooover II[24] are derived from Bluetooth Hoover, because they run on handheld devices, such as PDAs or mobile phones, and are capable of stealing sensitive information by using a BlueBugging attack[21] [1]. A *Blooovering attack* [23,24] can be done secretly by using only a Java 2 Micro Edition (J2ME) compatible Bluetooth mobile phone or handheld device with Blooover or Blooover II installed. They are intended to serve as auditing tools, which can be used for checking whether Bluetooth devices are vul-

[21]http://trifinite.org/trifinite_stuff_bluebug.html
[22]http://www.saftware.de/bluetooth/btxml.c
[23]http://trifinite.org/trifinite_stuff_blooover.html
[24]http://trifinite.org/trifinite_stuff_bloooverii.html

nerable or not, but they can be used for attacking against Bluetooth devices as well.[23,24] [1]

A *HeloMoto attack* [25] is a combination of BlueSnarfing (see Section 3.1) and BlueBugging attacks. HeloMoto attack is based on the poorly implemented handling of the trusted devices on some Motorola's Bluetooth mobile phones (for example, models V80, V5xx, V6xx, and E398 are vulnerable without firmware/software update) and it gives full access to the GSM AT command set.[25] [18]

An *On-Line PIN Cracking attack* [6] [1] means that an attacker is trying to connect with the target device by guessing different PIN values. It is based on the idea of changing the BD_ADDR of the attacking device every time a PIN guess fails, i.e. the attacker bypasses the ever increasing delay between retries. An On-Line PIN Cracking attack works only when the target device has a fixed or short adjustable PIN code. We designed, implemented, and tested our own On-Line PIN Cracking Security Analysis Tools, *On-Line PIN Cracking Script* [1, 14] and *On-Line PIN Cracking Tool* [1, 17], which are (as far as we know) the only security analysis tools for an On-Line PIN Cracking attack implemented so far. A detailed description of our security analysis tools can be found in [1,14,17]. [6] [1]

Our Bluetooth security attack, a *BTKeylogging attack* [1, 14], extends both the Brute-Force BD_ADDR Scanning attack[6] [1] and On-Line PIN Cracking attack[6] [1]. A BTKeylogging attack is carried out on a wireless connection between a Bluetooth-enabled keyboard and a PC: in the attack, an attacker uses the target device (a Bluetooth-enabled keyboard) as a "Bluetooth keylogger" by intercepting all packets (i.e. all keypresses) sent via air and decrypting them. The attack is possible when the target keyboard has a fixed or short adjustable PIN code. Moreover, the attacker must witness the initial pairing process between the target keyboard and the target computer: thereafter, all intercepted information, such as all usernames, passwords, and sent e-mails, can be decrypted. A detailed description of our attack can be found in [1,14]. [1]

[25]http://trifinite.org/trifinite_stuff_helomoto.html

Our Bluetooth security attack, a *BTVoiceBugging attack* [1, 14], also extends both a Brute-Force BD_ADDR Scanning attack and an On-Line PIN Cracking attack. A BTVoiceBugging attack is possible when the target device has a fixed or short adjustable PIN code and it has support for SCO or eSCO links. In the attack, an attacker uses the target device (for example, a Bluetooth headset or a Bluetooth-enabled PC/laptop equipped with a microphone and speakers) as a "Bluetooth bugging device". In such a case, the attacker can listen to sensitive conversations (for example, important business or other meetings taking place in the vicinity of the target device) via a SCO or an eSCO link, and she can also record these conversations for later use. We also defined three different BTVoiceBugging attack scenarios in order to eavesdrop on a typical business meeting. A detailed description of our attack as well as its different scenarios can be found in [1, 14]. [1]

For example, BTPrinterBugging attacks and attacks based on using Bluetooth worms/viruses can be classified as *disclosure, integrity, and DoS threats*.

We designed, implemented, and tested our *BTPrinterBugging attacks* [1, 19], which are based on the idea that an attacker abuses the target Bluetooth-enabled printer in order to do various harmful things: the attacker can, for example, both intercept and decrypt all the information that is sent to the printer (a *BTPrinterBugging via Packet Interception attack* [1, 19] can be performed using our *BTPrinterBugging via Packet Interception Security Analysis Tool* [1, 19]), use the printer remotely as if it was her own (a *BTPrinterBugging via Impersonation attack* [1, 19] can be performed using our *BTPrinterBugging via Impersonation Security Analysis Tool* [1,19]), deny access to the printer from the legitimate piconet users (a *BTPrinterBugging via Access Denial attack* [1, 19] can be performed in Windows environments by using our *BTPrinterBugging via Access Denial Security Analysis Tool* [1, 19] and in Linux environments by using our *BTPrinterBugging via Access Denial Security Analysis Tool II* [1, 19]), and do many other harmful things. A detailed description of our security analysis tools and attacks can be found in [1, 19]. [1]

Bluetooth worms and viruses have been often mentioned in the media, for example, in [26,27], because there are several Bluetooth worms and viruses, such as *Cabir* [28], *Skulls.D* [29], and *Lasco.A* [30], which use Bluetooth-enabled mobile phones for infecting other Bluetooth mobile phones. In addition, in January 2005, Brazilian software developer Marcos Velasco released all source codes of his Lasco.A worm/virus on his homepage[31], but later on removed them. However, Lasco.A sources can still be downloaded from many Brazilian file servers. It means that practically anyone can now write their own Bluetooth viruses just by modifying Lasco.A sources. In addition, Bluetooth worms and viruses can be very dangerous if the target device is vulnerable to BlueBugging, because in that way an attacker can slip in a virus or worm without alerting the user. Moreover, it is expected that attackers will exploit the techniques for finding hidden Bluetooth devices in an average of one minute[10] [15] in order to spread viruses and worms more efficiently. [1]

4 MITM ATTACKS ON BLUETOOTH

Our MITM attacks on SSP, *BT-Niño-MITM attack* [1,6], *BT-SSP-OOB-MITM attack* [1,8], *BT-SSP-Printer-MITM attack* [1,7], and *BT-SSP-HS/HF-MITM attack* [1,8] as well as the *SSP MITM attack of Suomalainen et al.* [5], are described in Section 4.1. Section 4.2 provides a literature review based comparative analysis of the existing MITM attacks on Bluetooth over the past ten years (2001-2011), including our MITM attacks on Bluetooth SSP.

[26]http://www.dailywireless.org/2005/02/04/bluetooth-viruses
[27]http://www.viruslist.com/en/analysis?pubid=204791928
[28]http://www.f-secure.com/v-descs/cabir.shtml
[29]http://www.f-secure.com/v-descs/skulls_d.shtml
[30]http://www.f-secure.com/v-descs/bluetooth-worm_symbos_lasco_a.shtml
[31]http://www.velasco.com.br

4.1 MITM attacks on SSP

We call our first attack as *BT-Niño-MITM attack* [1,6] (also referred to as *Bluetooth - No Input, No Output - Man-In-The-Middle attack*). In the attack we exploit the fact that the devices must exchange information about their IO capabilities during the first phase of the SSP (see Section 2). The exchange is done over an unauthenticated channel, and an attacker that controls this channel can therefore modify the information about capabilities and force the devices to use the association model of her choice. In our attack, the devices are forced to use the Just Works association model, which does not provide protection against MITM attacks. The MITM uses two separate Bluetooth devices with adjustable BD_ADDRs for the attack. Such devices are readily available on the market. The MITM clones the BD_ADDRs and user-friendly names of the victim devices in order to impersonate them more plausibly. We also described three general scenarios for the attack. A detailed description of our attack and its scenarios can be found in [1,6]. [1]

Suomalainen et al. [5] have presented an attack against SSP similar to our BT-Niño-MITM attack. In their attack the MITM prompts one device to use the normal Numeric Comparison association model, while forcing the other device to use the insecure Just Works association model. This leads to one of the devices (the one which uses the Numeric Comparison association model) treating the resulting link key as authenticated, and it might choose to trust it even for an application which requires a high level of security. However, this attack looks somewhat suspicious from the point of view of the user: one of the devices asks the user to compare the integrity checksums, while the other device does not display any numbers. In the tests performed by Suomalainen et al. [5], only 6 users out of 40 accepted the pairing on both devices. *Compared with the SSP MITM attack of Suomalainen et al. [5], our BT-Niño-MITM attack looks less dubious:* indeed, the user is only asked to confirm the pairing on both devices by pressing a button. Moreover, since this confirmation request is optional in the Bluetooth specification[1], some of the manufacturers might choose to skip it in order to improve usability. [1,6]

We call our second attack as *BT-SSP-OOB-MITM attack* [1, 8] (also referred to as a *Bluetooth - Secure Simple Pairing - Out-Of-Band - Man-In-The-Middle attack*). The attack requires that an attacker can somehow see the victim devices, i.e. there must be some kind of visual contact (for example, a hidden video camera or direct line-of-sight) to the victim devices. In the attack legitimate users are misled to select a less secure option instead of using a more secure OOB channel, for example, USB cable, Infrared Data Association (IrDA), or NFC. The attack works against any two OOB-capable Bluetooth devices that support SSP. We also described a general scenario for the attack. A detailed description of our attack and its scenario can be found in [1,8]. [1]

We call our third attack as *BT-SSP-Printer-MITM attack* [1, 7] (also referred to as a *Bluetooth - Secure Simple Pairing - Printer - Man-In-The-Middle attack*). In this attack we exploit the fact that almost all Bluetooth-enabled printers that support SSP (especially those connected using Bluetooth USB printer adapters) will use the Just Works association model in order to make printing user-friendly. It is not likely that users will be required to press any printer buttons just to accept the connection establishment in the initial pairing process of SSP. Therefore, the Just Works association model seems to be the most logical choice for SSP-enabled printers. Our attack works even against such SSP-enabled printers that should provide MITM protection via the Numeric Comparison, the Passkey Entry, or the OOB association model, because victim devices can be forced to use any association model that the attacker chooses [1,5–9]. We also described two scenarios for the attack. A detailed description of our attack and its scenarios can be found in [1,7]. [1]

We call our fourth attack as *BT-SSP-HS/HF-MITM attack* [1, 8] (also referred to as a *Bluetooth - Secure Simple Pairing - Headset/Hands-Free - Man-In-The-Middle attack*). In the attack we exploit the fact that almost all Bluetooth-enabled headsets and hands-free devices that support SSP will use the Just Works association model in order to make pairing process user-friendly. Our attack works even against such SSP-enabled headsets and hands-free devices that should provide MITM protection via the Numeric Comparison, the Passkey Entry, or the OOB association model, because victim devices can be forced to use the Just Works association model [1,5–9]. We also described a scenario for the attack. A detailed description of our attack and its scenario can be found in [1,8]. [1]

After a successful BT-Niño-MITM attack, BT-SSP-OOB-MITM attack, BT-SSP-Printer-MITM attack, BT-SSP-HS/HF-MITM attack, or SSP MITM attack of Suomalainen et al., the MITM can intercept and modify all data exchanged between the victim devices, and even use certain services that victim devices offer. [1,5–9]

4.2 Comparative Analysis of Bluetooth MITM Attacks

The first MITM attack on Bluetooth was devised in 2001 by Jakobsson and Wetzel [2]: the attack is also world's first Bluetooth security attack and thus *the year 2011 is a major 10-year milestone in the history of Bluetooth security attacks* – this milestone was also a big motivator in writing this research paper.

Even though the attack was devised for the version 1.0B of the Bluetooth standard, it works also with all Bluetooth versions up to 2.0+EDR. The attack assumes that the link key used by two victim devices is known to the attacker. The authors also showed how to obtain the link key using an Off-Line PIN Recovery attack (see Section 3.1). The MITM attack requires that both devices are in public or private security level (see Section 2), i.e. both victim devices must be connectable. In the attack, the BD_ADDRs of the attacker's devices must be cloned to equal the addresses of the victim devices. Moreover, to prevent the jamming of the communication channel, the victim devices must be both masters or both slaves (in two different piconets). In this case they transmit in unsynchronized manner and cannot see the messages of each other, while communicating with the attacker. After establishing connection to both victims, the attacker sets up two new link keys. [1,2,7,9]

Kügler [3] further improves the attack of Jakobsson and Wetzel. By manipulating with the clock settings, the attacker forces both victim devices to use the same channel hopping sequence but different clocks. In this way, the victim devices are unsynchronized and can see only the messages the attacker sends them. Moreover, Kügler [3] shows how a MITM attack can be performed during the paging procedure. The attacker responds to the page request of the master victim faster than the slave victim and restarts the paging procedure with the slave using a different clock. The master and slave use the same channel hopping sequence, but a different offset in this sequence. The attack works also in case when both victim devices send and receive data packets over an encrypted link. Even though the Initialization Vector (IV) used for encryption depends on the clock, the last bit of the clock is unused. Therefore, the attacker can flip this last bit, forcing the victims to use clocks which have the difference of approximately 11.65 hours. Although the integrity of data is protected with Cyclic Redundancy Checks (CRCs) which are appended to the plaintext prior to encryption, the attacker can manipulate intercepted ciphertext. After modifying the ciphertext in a certain way, the attacker updates the CRC bits (see [20] for details): the integrity checks performed by the victims do not detect the modification. It must be noted, however, that the attacker does not have much time for manipulating the transmitted data. [1,3,7,9]

As we discussed in Section 3.2, *Reflection attacks* [4] (also referred to as *Relay attacks*) can also be seen as a type of a MITM attack against authentication, but not encryption. The only information needed is the BD_ADDRs of the victim devices. During the paging procedure, the attacker responds to the request of the first victim device (A), and initiates a connection to the second victim device (B), posing as A. If the victim devices can hear each other, the mechanisms described in [3] can be used to achieve this. In the attack, the messages are simply relayed by the attacker's devices. The attacker can successfully perform authentication by using reflection attacks, but she cannot continue the attack if the target devices encrypt their communication. By combining reflection attacks with a known secret PIN code, link key, or encryption key, the attacker can both impersonate the victim devices and decrypt the information transferred between them. Victim devices can detect the attack by noticing a considerable increase in latency of getting the response to authentication challenge, caused by relaying. This countermeasure is not described in the standards, and it is up to the discretion of manufacturers to provide it. [1,3,4,7,9]

The versions 2.1+EDR, 3.0+HS, and 4.0 of Bluetooth provide protection against the abovementioned MITM attacks, by the means of SSP described in Section 2. However, it has been shown that MITM attacks against SSP-enabled Bluetooth devices are also possible by forcing victim devices to use the Just Works association model [1,5–9] (see Section 4.1). Thus, the attacker can bypass all security checks which would normally be in place. The association is then unauthenticated: the devices are aware of this fact, but it depends on the manufacturer how they react to this. If the victim devices have already established a link key, the attacker can use jamming to disrupt the communication and then she can initiate the connection under a chosen association model with both devices. As a result, the attacker learns the link key used by the devices and she can intercept all data transmitted between the devices. [1,7,9]

In Table 1 we summarize the properties of the MITM attacks overviewed in this section. It is interesting to note the connection of MITM attacks to other developments in the Bluetooth security analysis. For instance, at the time when most of the MITM attacks were introduced, implementing them was not an easy task, as there were no devices with adjustable BD_ADDRs, except sophisticated and expensive protocol analyzers. Now the situation has changed: Bluetooth devices with an adjustable BD_ADDR are readily available and techniques for finding hidden (non-discoverable) Bluetooth devices have been invented (see Section 3.1). Therefore, the danger of MITM attacks has recently increased. [1,7,9]

Table 1: MITM Attacks on Bluetooth: a Summary and Comparison. [1,7,9]

Attack:	[2]:	[3]:	[4]:	[5]:	[6–8]:
Bluetooth versions:	1.0–2.0+EDR	1.0–2.0+EDR	1.0–2.0+EDR	2.1+EDR–4.0	2.1+EDR–4.0
Attack goals:	Impersonation, modification	Impersonation, modification	Impersonation	Impersonation, modification	Impersonation, modification
Attacking devices:	2	2	2	2+1[32]	2+1[32]
Devices attacked:	Connectable	Connectable or non-connectable	Connectable or non-connectable	Connectable or non-connectable	Connectable or non-connectable
Distances:	Any[33]	Any[33]	Any[33,34]	Any[33]	Any[33]
Detection:	By user[35]	None[36]	By device[37]	By user[38]	By user[39]
Main countermeasure:	Security policies[40]	Integrity checks[41]	Detecting the delays	At the user interface level	At the user interface level[42]

5 NEW PRACTICAL COUNTERMEASURE FOR SSP

The simplest and cheapest countermeasure against SSP MITM attacks is to enforce devices to accept only authenticated link keys. Thus, *we propose that the devices should require a MITM protection during SSP and enforce it by not accepting unauthenticated link keys*, which are generated only by the Just Works association model. In practice, this can be accomplished as follows. The Bluetooth specification[1] discusses the concept of a "security database" that contains an entry for each service along with the security requirements of that

[32]A jamming device is also required.

[33]The attacker must use two Bluetooth adapters. Actual distance is limited by speed of the link between the attacker's devices.

[34]The victim devices must be out of each other's range.

[35]The user enters PIN to renegotiate.

[36]The attack remains undetected.

[37]There are delays in getting the LMP authentication response.

[38]One of the devices asks to compare numbers, while the other one does not.

[39]No Numeric Comparison is used although both devices have displays and keyboards.

[40]Security policies protecting against MITM attacks are proposed.

[41]Cryptographic integrity checks of packets should be used.

[42]Modifications to SSP specification are also proposed.

service. Bluetooth protocol stacks commonly include this ″security database″ function. One of these security requirements should be whether the device requires an authenticated or unauthenticated link key. If this security requirement is not met, access to the service is not granted. Therefore, this simple countermeasure is up to the discretion of Bluetooth protocol stack provider, i.e. the countermeasure can be used only if the Bluetooth protocol stack provides a ″security database″ function.

Other countermeasures against SSP MITM attacks are described in [1,6–9]. In addition, various Bluetooth security attacks in progress can be prevented and stopped by monitoring communication to discover such attacks: thus, we have proposed an *Intrusion Detection and Prevention System* [1,21] for Bluetooth networks to prevent and stop attacks in progress. A detailed description of our proposal can be found in [1,21]. Moreover, we feel that the use of *RF fingerprints* (also referred to as *RF signatures*) [22–24] could be the future of secure Bluetooth communications: therefore, we have proposed an *RF Fingerprint-Based Security Solution for SSP* [22]. A detailed description of our proposed system can be found in [22]. Furthermore, a detailed description of countermeasures for Bluetooth devices up to 2.0+EDR are described in [1,16,17].

6 NEW PRACTICAL ATTACK

Based on our findings and practical experiments on Big NAK attacks [1, 17] as well as our research work conducted on proposing an Intrusion Detection and Prevention System for Bluetooth networks [1, 21], we propose a new attack called *Big POLL attack*, which works against all existing Bluetooth versions, i.e. Bluetooth versions 1.0A – 4.0. In the attack, the attacker keeps victim devices (piconet slaves) busy all the time by sending repeated POLL packets to them so that they will not go into sleep or low-power mode. The Big POLL attack is possible, because during a normal piconet operation, the master device can use POLL packets to check that slave devices are still alive (i.e. up and running), and slave devices must always respond to a POLL packet sent by the master device. The attacker needs to discover the BD_ADDRs of all piconet devices (i.e. the BD_ADDR of master device and BD_ADDRs of slave devices), impersonate the piconet master (i.e. duplicate its BD_ADDR), and start sending POLL packets to all piconet slaves.

Let us assume the following attack scenario:

1. The attacker uses a Brute-Force BD_ADDR Scanning attack, a Bluetooth protocol analyzer, techniques for finding hidden Bluetooth devices in an average of one minute, or 79 Bluetooth receivers in parallel to discover the hidden (non-discoverable) BD_ADDRs of the piconet master and all piconet slaves.

2. The attacker impersonates the piconet master by duplicating its BD_ADDR.

3. The attacker starts sending POLL packets non-stop to all piconet slaves, thus keeping them busy all the time.

In our attack, the attacker consumes the batteries of the victim devices (piconet slaves) rather quickly. Moreover, piconet slaves are not getting the legitimate piconet services within a reasonable time, because the attacking device keeps them busy all the time. It is worth noting that the attacker does not need to witness the initial pairing process between the victim devices or intercept any random numbers sent via air: the only information required is the BD_ADDRs of the victim devices.

Big POLL attacks can be very annoying if the attacker uses them non-stop to deny the legitimate piconet devices access to the piconet services. Moreover, the attacker can use these kinds of attacks to mislead the target devices in such a way that they delete previously stored link keys so that the initial pairing process is restarted.

The only feasible countermeasure for a Big POLL attack seems to be the following: *a Bluetooth network should use our Intrusion Detection and Prevention System* [1, 21] or some other intrusion detection and prevention system that is capable of detecting a Big POLL attack.

7 CONCLUSION AND FUTURE WORK

A literature review based comparative analysis of Bluetooth security attacks over the past ten years (2001-2011) was provided in the paper. In addition, a new practical countermeasure against MITM attacks on SSP was proposed. Moreover, a novel attack that works against all existing Bluetooth versions was proposed.

The current level of security is insufficient in many Bluetooth devices on the market as this research paper clearly shows. Since there are billions of Bluetooth devices in use without SSP's improved security features, malicious security violations are not expected to decrease in the near future. On the contrary, these old Bluetooth devices will be sold for many years to come, thus making security concerns even more alarming. Furthermore, MITM attacks on SSP are also possible by forcing victim devices to use the Just Works association model.

Since we have now covered all noteworthy Bluetooth security attacks, including our own attacks, during the past ten years (2001-2011) in this paper, it is also valuable to summarize the dangerousness and practical relevance of the attacks: thus, Figure 1 summarizes all attacks described in this paper by providing dangerousness and practical relevance figures.

The problems we want to investigate in our future research work are concerned with the following issues:

1. Since we have already proposed an Intrusion Detection and Prevention System for Bluetooth networks as well as an RF Fingerprint-Based Security Solution for SSP, we want to combine these research results by implementing a working prototype of such a combination system and also analyze its efficiency.

2. Since it is nowadays possible to acquire the hardware required for MITM attacks, we want to make practical implementations of all existing Bluetooth MITM attacks. Moreover, we want to analyze the results of the practical experiments, draw conclu-

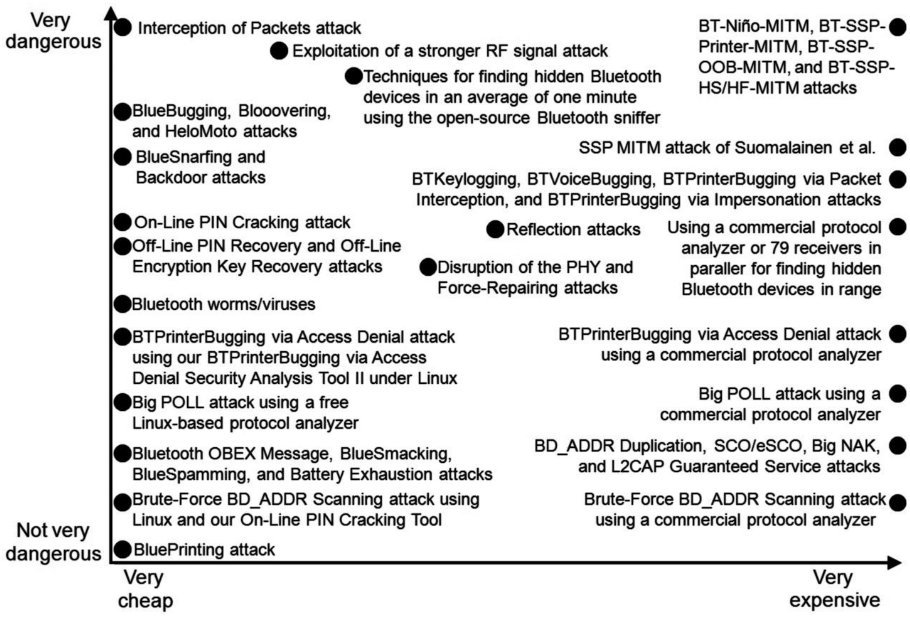


Figure 1: Our estimation of dangerousness and practical relevance of the attacks.

sions, and propose practical countermeasures based on our findings.

3. Since there are many new emerging wireless technologies, such as ZigBee and Ultra-Wideband (UWB), which are quite similar to Bluetooth technology, it is expected that our Bluetooth security related research work can be quite easily extended to cover the security of these new technologies. Therefore, we want to investigate how various Bluetooth security attacks and their countermeasures can be ported to support ZigBee and UWB technologies. In fact, we are currently conducting a practical research work on ZigBee security attacks.

4. We feel that the use of steganography [25–28] could be one potential solution for securing Bluetooth communications in the future: in fact, we have performed some research work on steganography [25, 26] and we are also currently conducting

a research work on the computational aspects of watermarking and steganography for allowing secure authentication between the communicating Bluetooth devices, i.e. we plan to embed certain messages into digital images by using digital watermarking and/or steganography that hides the existence of the messages allowing secure extraction of these embedded messages only by the legitimate recipient.

Acknowledgements

We gratefully acknowledge Elena Trichina, Martti Penttonen, and Tapio Grönfors for their guidance and supervision during the work presented in Haataja's Ph.D. thesis [1]. We also thank Sanna Pasanen and Niina Päivinen for their cooperation on RF Fingerprint related research work [22].

References

[1] K. Haataja, *Security Threats and Countermeasures in Bluetooth-Enabled Systems*, Ph.D. Diss., University of Kuopio, Department of Computer Science, Feb. 6, 2009.

[2] M. Jakobsson and S. Wetzel, "Security Weaknesses in Bluetooth," *Lecture Notes in Computer Science*, Vol. 2020, pp. 176–191, Springer-Verlag, 2001.

[3] D. Kügler, "Man-In-The-Middle Attacks on Bluetooth," *Lecture Notes in Computer Science*, Vol. 2742, pp. 149–161, Springer-Verlag, 2003.

[4] A. Levi, E. Cetintas, M. Aydos, C. Koc, and M. Caglayan, "Relay Attacks on Bluetooth Authentication and Solutions," *Lecture Notes in Computer Science*, Vol. 3280, pp. 278–288, Springer-Verlag, 2004.

[5] J. Suomalainen, J. Valkonen, and N. Asokan, "Security Associations in Personal Networks: A Comparative Analysis," *Lecture Notes in Computer Science*, Vol. 4572, pp. 43–57, Springer-Verlag, 2007.

[6] K. Hyppönen and K. Haataja, ""Niño" Man-In-The-Middle Attack on Bluetooth Secure Simple Pairing," in *Proceedings of the Third IEEE International Conference in Central Asia on Internet, The Next Generation of Mobile, Wireless, and Optical Communications Networks*, Tashkent, Uzbekistan, Sep. 26–28, 2007.

[7] K. Haataja and K. Hyppönen, "Man-In-The-Middle Attacks on Bluetooth: a Comparative Analysis, a Novel Attack, and Countermeasures," in *Proceedings of the Third IEEE International Symposium on Communications, Control, and Signal Processing*, St. Julians, Malta, Mar. 12–14, 2008.

[8] K. Haataja and P. Toivanen, ”Practical Man-In-The-Middle Attacks Against Bluetooth Secure Simple Pairing,” in *Proceedings of the 4th IEEE International Conference on Wireless Communications, Networking, and Mobile Computing*, Dalian, China, Oct. 12–14, 2008.

[9] K. Haataja and P. Toivanen, ”Two Practical Man-In-The-Middle Attacks on Bluetooth Secure Simple Pairing and Countermeasures,” *IEEE Transactions on Wireless Communications*, Vol. 9, No. 1, pp. 384–392, Jan. 2010.

[10] R. Morrow, *Bluetooth: Operation and Use*, New York, USA, McGraw-Hill, 2002.

[11] Y. Shaked and A. Wool, ”Cracking the Bluetooth PIN,” in *Proceedings of the 3rd ACM International Conference on Mobile Systems, Applications, and Services*, Seattle, Washington, USA, Jun. 6–8, 2005, pp. 39–50.

[12] C. Gehrmann, J. Persson, and B. Smeets, *Bluetooth Security*, Boston, Massachusetts, USA, Artech House, 2004.

[13] J. Massey, G. Khachatrian, and M. Kuregian, ”SAFER+,” in *Proceedings of the First NIST Advanced Encryption Standard Candidate Conference*, Ventura, California, USA, Aug. 20–22, 1998.

[14] K. Haataja, ”Two Practical Attacks Against Bluetooth Security Using New Enhanced Implementations of Security Analysis Tools,” in *Proceedings of the IASTED International Conference on Communication, Network, and Information Security*, Phoenix, Arizona, USA, Nov. 14–16, 2005, pp. 13–18.

[15] D. Spill and A. Bittau, ”BlueSniff: Eve Meets Alice and Bluetooth,” in *Proceedings of the First USENIX Workshop on Offensive Technologies*, Boston, Massachusetts, USA, Aug. 6, 2007.

[16] K. Haataja, ”Bluetooth Security Threats and Possible Countermeasures,” in *Proceedings of the Annual Finnish Data Processing Week at the University of Petrozavodsk on Advances in*

Methods of Modern Information Technology, Petrozavodsk, Russia, 2005, Vol. 6, pp. 116–150.

[17] K. Haataja, ”Three Practical Bluetooth Security Attacks Using New Efficient Implementations of Security Analysis Tools,” in *Proceedings of the IASTED International Conference on Communication, Network, and Information Security*, Berkeley, California, USA, Sep. 24–26, 2007, pp. 101–108.

[18] K. Haataja, ”Bluetooth Network Vulnerability to Disclosure, Integrity, and Denial-of-Service Attacks,” in *Proceedings of the Annual Finnish Data Processing Week at the University of Petrozavodsk on Advances in Methods of Modern Information Technology*, Petrozavodsk, Russia, 2006, Vol. 7, pp. 63–103.

[19] K. Haataja, ”New Practical Attack Against Bluetooth Security Using Efficient Implementations of Security Analysis Tools,” in Proceedings of the IASTED International Conference on Communication, Network, and Information Security, Berkeley, California, USA, Sep. 24–26, 2007, pp. 134–142.

[20] N. Borisov, I. Goldberg, and D. Wagner, ”Intercepting Mobile Communications: the Insecurity on 802.11,” in Proceedings of the 7th ACM Annual International Conference on Mobile Computing and Networking, Rome, Italy, Jul. 16–21, 2001, pp. 180–189.

[21] K. Haataja, ”New Efficient Intrusion Detection and Prevention System for Bluetooth Networks,” in *Proceedings of the ACM International Conference on Mobile, Wireless MiddleWare, Operating Systems, and Applications*, Innsbruck, Austria, Feb. 12–15, 2008.

[22] S. Pasanen, K. Haataja, N. Päivinen, and P. Toivanen, ”New Efficient RF Fingerprint-Based Security Solution for Bluetooth Secure Simple Pairing,” in *Proceedings of the 43rd IEEE Hawaii International Conference on System Sciences*, Koloa, Kauai, Hawaii, Jan. 5–8, 2010.

[23] M. Barbeau, J. Hall, and E. Kranakis, ”Detecting Impersonation Attacks in Future Wireless and Mobile Networks,” *Lecture Notes in Computer Science*, Vol. 4074, pp. 80–95, Springer-Verlag, 2006.

[24] O. Ureten and N. Serinken, ”Wireless Security Through RF Fingerprinting,” *Canadian Journal of Electrical and Computer Engineering*, Vol. 32, No. 1, pp. 27–33, 2007.

[25] A. Kaarna and P. Toivanen, ”Digital Watermarking of Spectral Images in PCA/Wavelet-Transform Domain,” in *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium*, Toulouse, France, Jul. 21–25, 2003, Vol. 6, pp. 3564–3567.

[26] A. Kaarna, P. Toivanen, and K. Mikkonen, ”Watermarking Spectral Images Through the PCA Transform,” in *Proceedings of the IS&T Image Processing, Image Quality, and Image Capture Systems Conference*, Rochester, New York, USA, May 13, 2003, Vol. 6, pp. 220–225.

[27] C. Podilchuk and E. Delp, ”Digital Watermarking: Algorithms and Applications,” *IEEE Signal Processing Magazine*, Vol. 8, No. 4, pp. 33–46, Jul. 2001.

[28] D. Artz, ”Digital Steganography: Hiding Data Within Data,” *IEEE Internet Computing*, Vol. 5, No. 3, pp. 75–80, May/Jun. 2001.

Non-repudiation and Smart Cards

Marko Hassinen
University of Eastern Finland
School of Computing
P.O. Box 1627, 70211 Kuopio, Finland

Abstract. Moving from paper documents and written contracts into the era of electronic transactions and agreements, there is a major hurdle with assuring authenticity of a contract and commitment of the parties. This paper discusses the concept of digital signature and some technical aspects of modern digital signature schemes. In particular, smart cards and their usability in deriving unforgeable and non-repudiable digital signatures are discussed.

1 INTRODUCTION

Since ancient times people have had a need to make contracts, and, also derive a document for later reference. In order to mitigate a future dispute concerning the content of an agreement, there has generally been two or more identical copies of a contract document. Furthermore, for assuring genuinity of such a document, means of authenticity are probably as old as first written contracts. Before writing came a widely common skill, some kind of drawn personal marks were used. Traditional signatures are obviously handwritten.

Quite often with written documents there are two independent persons who vouch for the authenticity of the signature. At the same time these persons generally attest that a signature has been made willingly and not under any kind of force. Should there be a subsequent dispute about the content of the contract or the authenticity of a document, these people can be consulted. In case the contract was signed without independent witnesses, methods of signature authenticity verification can be enforced.

A digital signature is an attestation of some digital content. Most often it is used to show commitment to a digital document.

The most general scenario is to use a fixed-length hash value (you can think of it as a type of ”fingerprint”) of the document that is subsequently encrypted using the signer's private encryption key.

Traditional handwritten signatures also have other important properties that are just as important when considering a digital signature. A handwritten signature can not be detached from the document and transferred to another one. This seems trivial but considering the case of digital signatures, detaching the signature from the document is really easy, and most often these are separate by default. This is not an actual problem as long as the signature cannot be attached to another, different document.

2 HOW DIGITAL SIGNATURES ARE MADE

To create a digital signature that has the same legal force as a written signature on a paper document, one has to consider the properties of handwritten signatures and match or exceed them. The main properties are resistance to forgery, unquestionable binding to the document which is signed and non-repudiation.

The first two properties are obtained through using hash algorithms. A hash algorithm is an algorithm that takes a variable size input and produces a fixed size output. The input of a hash algorithm is called a preimage. If two distinct preimages produce a common result, there is a collision. A good hash algorithm has strong collision resistance, which means that it is infeasible to find two distinct preimages that produce the same hash result. There is also weak collision resistance in which it is infeasible to find an input that produces the same hash as a given input.

The hash itself is not adequate digital signature as it is trivial for an attacker to change the document and calculate a new hash. This corresponds to a situation with paper documents where it would be enough to have a signature on a contract regardless whose signature that would be. In order to protect the hash, an additional security measure, namely encryption, has to be applied. To protect the hash from tampering and tying it to the signer, Public Key

Cryptography is used.

Public Key Cryptography is a form of asymmetric encryption and relies on a concept of two different keys, hence a key pair. These keys are mathematically related but it is not feasible to cover one key just by knowing the other. Generally the keys are referred to as a public key and a private key. Clearly, the private key is meant to be held private and it should be known only to the owner of the key. In certain cases even the owner does not know the private key, but is in possession of the key and can use it for cryptographic operations. Probably the most used public key algorithm is RSA (Rivest-Shamir-Adleman) that was invented 1977 by the aforementioned.

Asymmetric encryption refers to a method in which a key pair is used and the key pair works in the following way; if one key is used for encryption, the other one can be used for decryption. Furthermore, each key has only one unique counterpart and encryption done with one key can only be decrypted using the other one. A digital signature is obtained by encrypting the hash value using the private key of the signer. Assuming the key is truly private, no-one else can produce such a signature. Figure 1 shows a crude outline of how sender (Alice) uses a hash algorithm to derive a hash and her public key (in this case located on a smart card) to create a digital signature. She then sends this signature along with the document to Bob.

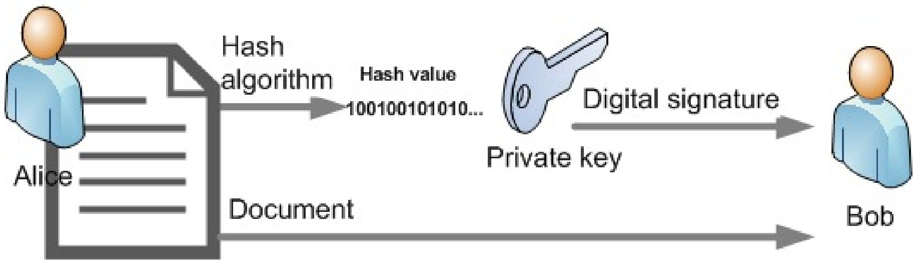


Figure 1: Basic description of a digital signature creation

The public key is, as the term implies, meat to be public and known to a wide audience. Communicating with a person or system that uses Public Key Cryptography, obtaining the correct public key is vital. To securely manage the mass of public keys, a system called Public Key Infrastructure (PKI) has been created. To understand the need for such a system, it is enough to imagine a situation where one has received a contact from a previously unknown source that uses Public Key Cryptography to protect the communication. The question arises where can one find the public key of the sender?

PKI system is as the name states an infrastructure for storing public keys. Without taking any stance on the architecture how the keys are stored, there is the issue of how to verify that a key belongs to the person/organisation it is claimed to belong to. Consider the case where the keys would be in a public database and one would retrieve a public key from the database. Should the keys have no protection against tampering, a man-in-the-middle (MTM) attack against communication would be trivial. Imagine a situation where you would use your private key to encrypt a message and send it to a person who does not already have your public key. This person would retrieve the key from a database and decrypt your message. In case the decryption worked, the recipient would be in (dis)belief that the message in fact came from you. Now, an attacker would intercept your message and decrypt the message with your key obtained from the database. Then the attacker would modify the message at will and encrypt the message with the his private key and send the message to the final recipient. Upon receiving the message, the recipient would query the public key database for your key. At this point the attacker would intercept the key query and answer with a message containing his public key. This would make the recipient believe that the message is genuine and originated from you.

In order to protect the public keys from forgery, a system of public key certificates has been introduced. The basic idea of a certificate is to hold the public key and relevant information, such as details of the key owner. The certificate itself is then digitally signed by an organisation called Certificate Authority (CA). The role of a CA is to grant public key certificates for individuals and/or organisations. In order to grant a certificate, it is the responsibility of the CA to verify by various means the true identity of the certificate applicant.

Upon receiving a digital signature, the recipient retrieves the signers public key certificate from the certificate repository (usually using a protocol called LDAP, Light Weight Directory Protocol) and verifies that the certificate is valid and genuine. In order to validate genuinity, the recipient needs to have the public key of the CA, which is also contained in a certificate signed by the CA itself. In Figure 2, Bob has received a document from Alice with adigital signature. To verify the signature, Bob retrieves Alices public key certificate from a database. With this public key Bob can decrypt the signature and obtain a hash value. Accordingly Bob can calculate a hash from the document he has received and by comparing these two hash values, Bob can determine if the signature is valid.

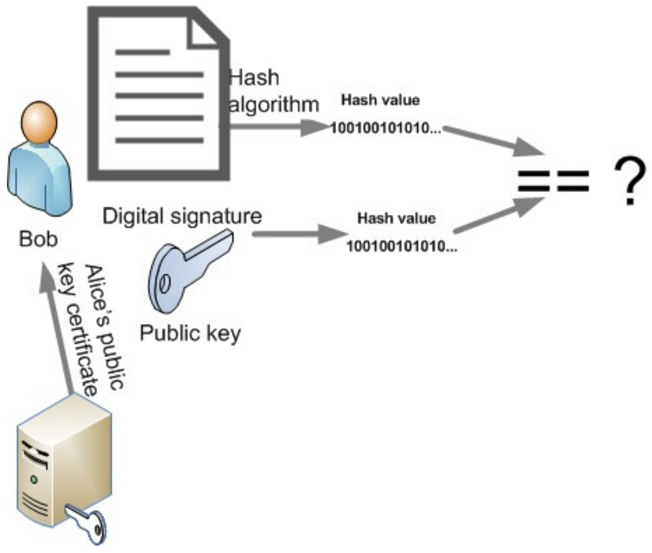


Figure 2: Basic description of a digital signature verification

Usually, computer systems come with a list of most common CA certificates and in case CA certificate is not on that list, the user can import a CA certificate into the certificate store. Obviously, in this case it is very important to verify that the CA certificate is obtained from a reliable source.

To assure interoperability with different environments, certificates are standardized in the X 509 standard. The X509 format contains most importantly the public key and owner information. The listing below shows content of one certificate and the details can be studied from the example.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            b4:86:40:84:96:d2:3c:10:a0:6a:64:0c:e7:a9:a4:26
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=FI, O=Sonera, CN=Sonera Class2 CA
        Validity
            Not Before: Oct 23 14:26:21 2008 GMT
            Not After : Oct 23 14:26:21 2011 GMT
        Subject: C=FI, ST=Pohjois-Savo, L=Kuopio, O=Unicta Oy,
                 CN=*.unicta.fi
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:d9:8c:48:59:ca:fa:f3:fb:f9:8c:8a:8c:82:05:
                    85:8b:1c:a0:2a:35:3b:dc:fa:37:1c:20:bf:92:f9:
                    77:a9:08:c5:89:c3:72:29:44:bd:f0:b5:9a:b5:24:
                    1c:c0:18:41:e5:67:91:e9:71:ad:80:a5:3e:78:63:
                    13:3f:c9:af:56:27:4b:29:79:0b:e3:ba:fa:e1:dd:
                    b3:87:08:6e:e3:bd:f6:27:07:12:d7:34:42:b4:61:
                    85:00:2f:2a:5b:51:eb:6c:62:a5:36:93:44:f7:b7:
                    7e:9f:14:a7:66:52:41:6b:b9:32:81:a7:b6:15:22:
                    9c:8d:13:c3:c3:17:98:6a:a5
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Authority Key Identifier:
                keyid:4A:A0:AA:58:84:D3:5E:3C

            X509v3 Certificate Policies:
                Policy: 1.3.6.1.4.1.271.2.3.1.1.2

            X509v3 CRL Distribution Points:
                URI:ldap://194.252.124.241:389/cn=Sonera%20Class2%

20CA,o=Sonera,c=FI?certificaterevocationlist;binary

            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client
                Authentication
            X509v3 Subject Key Identifier:
                29:A2:9F:7E:FE:31:CF:5C:B5:E6:
                EB:31:01:31:43:EE:36:69:95:CC
    Signature Algorithm: sha1WithRSAEncryption
        1a:80:83:14:9d:7f:b2:56:45:64:bb:e2:6a:d3:2a:18:29:a5:
        8e:ba:d8:b8:4e:b8:07:ac:fb:b4:e8:ed:7e:39:6d:4a:ce:15:
        84:7f:19:7b:02:72:36:e6:92:a8:a1:25:44:5d:32:1e:42:4e:
        4c:a4:d7:ce:04:f2:31:65:ce:4d:39:70:b1:da:23:f8:1c:fa:
        c7:26:c9:1d:05:70:46:3d:45:c3:24:bb:4a:03:bd:bb:01:94:
        c2:9f:00:52:b9:57:70:80:3f:01:1e:f7:cb:c5:b7:57:77:33:
        79:56:56:0f:d7:c7:21:0e:5c:6f:d5:16:b5:78:d4:99:8c:27:
        e0:1d:cb:22:33:9b:9d:f4:60:b4:d9:8c:99:e8:54:a6:e3:bb:
        47:f7:2f:4c:43:06:d1:b4:5d:ae:77:04:fe:05:5b:0d:a0:51:
        ae:34:d8:81:fc:26:df:38:b7:a9:6d:19:1b:d0:9e:51:74:0c:
        72:46:03:87:82:88:d9:e2:c9:a1:42:49:ae:bf:44:dc:e0:8c:
        ce:33:5f:b4:3f:34:95:44:16:b8:6a:e6:b8:6c:7f:9d:e3:8e:
        48:d2:fe:89:7a:9c:6f:fc:ce:2d:c0:29:6c:df:02:a4:77:23:
        86:94:3b:25:20:18:f6:e3:71:fe:81:10:6f:dd:b8:32:97:33:
        90:8b:d9:4b
```

There is naturally always the possibility that the private key corresponding to a public key on the certificate goes missing or gets compromised. In such case there is a need to void the certificate. This can be done using a Certificate Revocation List (CRL) that contains certificates that have been revoked for any reason. CRLs are updated periodically and in business scenarios where real time revocation information is vital, one can use Online Certificate Status Protocol (OCSP) that allows the recipient to query the status of a certificate.

DSS, Digital Signature Standard by National Institute of Standards and Technology (NIST) is the de-facto standard for Digital signatures. The most recent version was defined in 2009 as FIPS 186-3. A standard is essential in order to make different systems using digital signature interoperate. The FIPS standard reads; A digital signature is computed using a set of rules and a set of parameters that allow the identity of the signatory and the integrity of the data to be verified. Naturally, one needs a system to agree on and communicate those rules and parameters.

3 SMART CARDS

A Smart Card is generally a plastic card with a chip in it. Most common sizes are the traditional credit card size and SIM (Subscriber Identity Module) cards. Smart Cards can be thought of as small computers that get their operating power though the visible contacts on the card surface. Figure 3 shows a couple of examples of smartcards with the smart card contact area showing yellow. There are also contactless smart cards, but in this paper we shall concentrate on the contact version only.

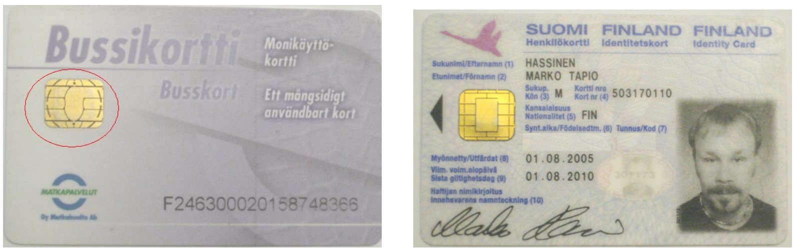


Figure 3: Some examples of smart cards

In addition to being able to make computations, smart cards contain variable amount of non volatile memory that can be used to store cryptographic keys and smart card applications. Many smart cards allow user to create custom made applications which can be then stored and run on the card.

For cryptographic purposes, smart cards have a very important feature called tamper resistance. Sometimes smart cards are said to be tamper proof, but there are different opinions on whether this assertion actually holds. Several attacks against smart card tamper resistance have been tried but with varying and usually not very admirable success. Most attacks have a questionable feasibility and are often too expensive to be practical.

The tamper resistant behaviour makes smart cards lucrative choice when storage of user credentials is an issue. Namely, storage of the private key in asymmetric encryption, authentication and digital signatures, can be problematic as the scenario really needs to keep the private key private. The benefit for using a smart card for storing a private key comes with the smart card ability to run applications on the card. This allows locating the key generation algorithm on the card and generating the keys on the card. Adding the feature of non volatile memory, we can come to a scenario in which the key pair is generated on the card and the private key can be kept on the card without ever bringing it outside the card.

Using such a card to create a digital signature would mean sending the hash value to be signed to the card and asking for the card to do the private key encryption. This way, a digital signature can be obtained without having to handle the private key. In essence most such cards are constructed in a way that does not allow the private key to be exported from the card, and this is a very important condition for obtaining non-repudiation. Naturally, the card is protected with a PIN (Personal Identification Number) that the user has to know to use the card.

After (or during) the key generation procedure the public key can obviously exported from the card. Most often the key generation happens on the initialization phase, when no information of the future user is present. Later on, as user data becomes available, the card is personalized at which point a public key certificate is created for the user and the visual appearance of the card is finalized.

4 CONCLUSION

Non-repudiation with digital signatures has a clear requirement that the signer has a secret that nobody else has. Non-repudiation can not be obtained using shared secrets. Smart cards are ideal for this purpose if we can be sure that a valid digital signature can only be created on the card. In doing so, we obtain a strong two-factor authentication, namely what the signer knows (the PIN) and what the signer has (the card). The fact that the secret key is located only on the card and can not be exported creates a situation in which we can conclude that a valid digital signature could not have originated from anywhere else the card.

Using a smart card to derive a digital signature has received the status of a legally binding show of commitment (assuming accredited PKI and CA). Such a commitment is valid in a court of law. However, one should realize that the system still is a TTP (Trusted Third Party) scenario in which a fraud is possible but requires a conspiracy of a large number of participants.

Further reading

1. DSS, Digital Signature Standard.
   http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf

2. Wolfgang Rankl,Wolfgang Effing: *Smart Card Handbook*

3. Bruce Schneier: *Applied Cryptography*

4. Menezes et al.: *Handbook of Applied Cryptography*

Matrix Based Calculation of All-Pairs Shortest Paths on the GPU

Risto T. Honkanen
Kajaani University Consortium, CEMIS-Oulu
P.O. Box 51
FIN-87101 Kajaani, Finland
Risto.Honkanen@oulu.fi,
http://cc.oulu.fi/∼rhonkane

Abstract. Calculation of large graph problems are common in many practical applications. Graphics Processing Units (GPUs) offer a large amount of parallelism and high computational power. In this study we present the all-pairs shortest-path problem and matrix based calculation of it on the GPU. We show that the GPU implementation is considerably faster that the CPU based if the sparseness of real world networks are not taken into account.

1 INTRODUCTION

Graph algorithms have a lot of use in practical solutions such as part of more complex algorithms or representing relationships occurring in many real-world systems. Those are, e.g., communication systems, electrical networks, data mining, transportation systems, or scheduling systems. Processing graphs efficiently has been a research focus for tens of years. Some of graph theory problems are, e.g., connected component problem (CCP), spanning tree problem (STP), and shortest path problem (SPP). In this presentation, we concentrate on all-pairs shortest path problem (APSP). Solving all-pairs shortest paths of a graph requires a large amount of computation. For instance, using the Floyd-Warshall algorithm to find APSPs for a weighted, directed graph requires $\Theta(|V|^3)$ time steps, where $|V|$ is the number of vertices in the graph [7].

In order to accelerate computation, a number of techniques have been presented, such as cluster and multi-core computing and graphics processing units (GPUs). The GPU initially was designed to accelerate graphics tasks in gaming and rendering [7]. The computing performance of GPUs has increased rapidly during the last few years. for example, NVIDIA offers a number of desk-top computers having hundreds of processing units at the graphics card(s) [5]. NVIDIA offers a programming model called Compute Unified Device Architecture (CUDA). CUDA allows programmers to use a unified programming framework and C-like subroutines to efficiently execute their parallel programs on a GPU.

Harish and Narayanan presents a number of CUDA based implementations of large graph algorithms in their paper [2]. They experimented algorithms designed for sparse matrices. According to their studies the GPU based algorithms is considerably faster when the degree of network rather high (6 – 7). Using real world data they found that the GPU is slower than CPU. According Harish and Narayanan this is because of the low average degree of the graphs [2]. Garland proposes the use of compressed sparsed row presentation in his paper [1]. This saves substantially the storage space and processing time [1].

In this paper, present a demonstration of matrix based calculation of APSP on the GPU. We show that the matrix based calculation of APSP can be implemented efficiently on the GPU in comparison to its serial counterpart. However, presenting real world routing problems by matrices usually leads to calculation of sparse matrices. This strongly reduces the efficiency of matrix based computing.

The paper is organized as follows. In Section 2 we present the CUDA programming model for GPUs. Section 3 discusses Warshall's transitive closure and Floyd-Warshall's algorithms. In Section 4 we present our implementation for calculation of APSPs on the GPU. Section 5 presents the results of our work. Finally, Section 6 discusses conclusions and future work.

2 CUDA PROGRAMMING ON THE GPU

A CUDA program consists of a serial part executed at the CPU and a number of kernels executed at the GPU. Section 2.1 introduces the the CUDA hardware model we use. In Section 2.2 we present the CUDA software model.

2.1 CUDA Hardware Model

At the hardware level, the graphics processing unit is a collection of a number of multiprocessors each having a number of processors or CUDA cores. A schema of CUDA hardware interface is presented in Figure 1.

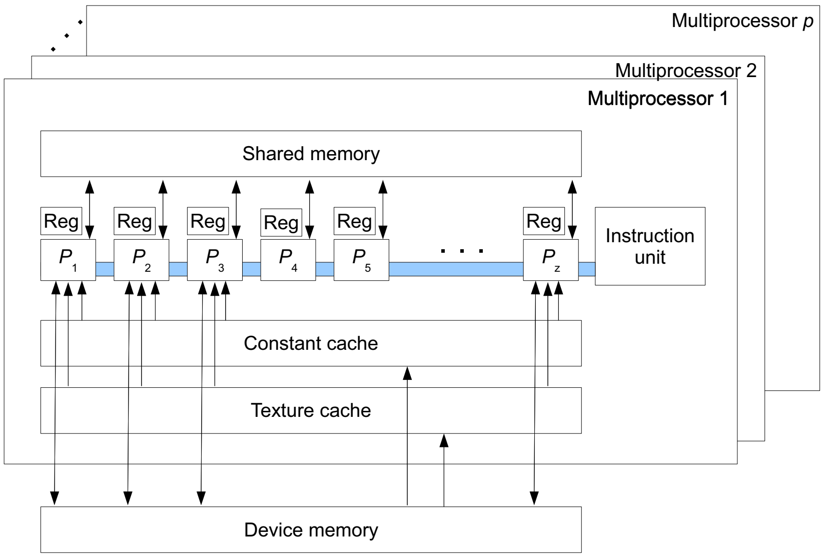


Figure 1: CUDA hardware model

The number of multiprocessors and CUDA cores vary a lot. Table 1 represents the number of multiprocessors and CUDA cores deployed at various NVIDIA GPU cards [6].

Table 1: Examples of NVIDIA's GPU cards.

	# of multiprocessors	# of CUDA cores
GeForce GTX 460	7	336
GeForce GTX 580	16	512
Tesla S 1070	4×30	4×240
Quadro FX 4800	24	192
Quadro FX 5600	16	128

Each multiprocessor has its own shared memory space such that CUDA cores deployed at the multiprocessor have equal access to the shared memory. Each core has its local register space. Multiprocessors and cores communicate with Central Processing Unit (CPU) via device memory space deployed at the graphics card.

2.2 CUDA Software Model

At the software level, the CUDA model consists of a number of independent *threads* running in parallel. The host side program usually consists serial part of the program executed at the CPU and a number of kernel programs executed at the GPU. A programmer usually

1. Decides how to divide the problem in independent sub-problems and forms a grid of blocks
2. Allocates enough memory space at the GPU for the kernel program
3. Transports the needed data to the GPU
4. Executes the kernel
5. Returns results at the memory space of the CPU
6. Deallocates memory space at the GPU

An schema of CUDA programming model is presented in Figure 2. On the left hand side of Figure 2 resizes the CPU part of the program. Right hand side of Figure 2 presents the GPU part of the program.

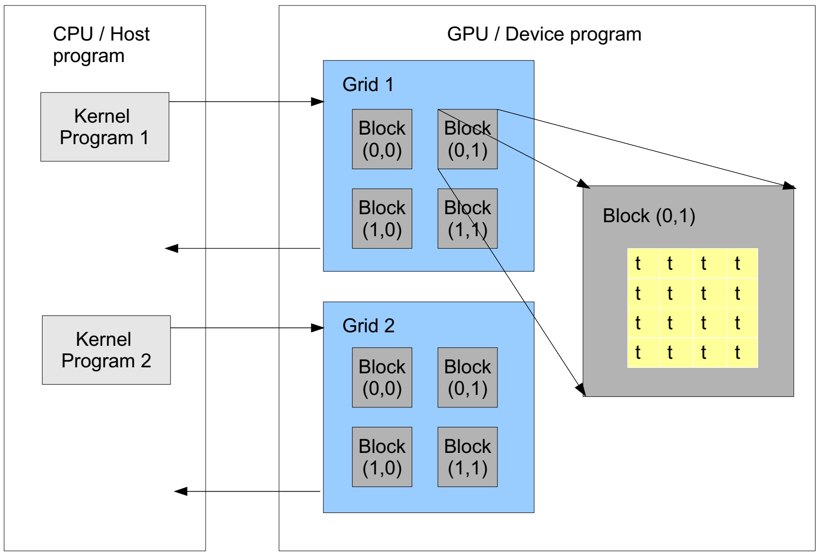


Figure 2: CUDA software model

The problem is divided in a grid of blocks. Each block consists of a number of independently executable threads. If multiple blocks are assigned to a single multiprocessor, their execution is time-shared. Each block has its own unique *block ID* and each thread inside a block has its own *thread ID*. The system allows the programmer access within any thread during its execution.

3 CALCULATING ALL-PAIRS SHORTEST PATHS

Calculating accessibility of nodes of a graph between each other is a basic problem with graphs. Warshall's transitive closure (WTC) algorithm is presented in Section 3.1. By modifying the WTC we are able to calculate all-pairs shortest-paths. Floyd-Warshall algorithm is presented in Section 3.2.

3.1 Warshall's Transitive Closure

Let us consider a directed graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. The transitive closure of G is defined as a graph $G+ = (V, E+)$ such that for all $v, w \in V$ there is an edge $(v, w) \in E+$ if and only if there is a non-null path from v to $w \in G$ [4]. Algorithm Transitive Closure (G) resolves the transitive closure of G [8].

```
Transitive Closure ($G$)
1: $n \leftarrow |V[G]|$
2: **for** $j \leftarrow 1$ to $n$
3:     **for** $i \leftarrow 1$ to $n$
4:         **if** $i = j$ or $(i, j) \in E[G]$
5:             **then** $t_{i,j}^{(0)} \leftarrow 1$
6:             **else** $t_{i,j}^{(0)} \leftarrow 0$
7: **for** $k \leftarrow 1$ to $n$
8:     **for** $i \leftarrow 1$ to $n$
9:         **for** $j \leftarrow 1$ to $n$
10:            $t_{i,j}^{(k)} \leftarrow t_{i,j}^{(k-1)} \vee (t_{i,k}^{(k-1)} \wedge t_{k,j}^{(k-1)})$
11:**Return** $T^{(n)}$
```

At the beginning of computation (on lines 1 – 6) transitivity table T is initialized. After that, transitivities of degree k are calculated by using transitivities of degree $k - 1$ and dynamic programming.

3.2 Floyd-Warshall Algorithm

We can solve the APSP using the Floyd-Warshall algorithm. Let us consider $G = (V, E)$ be a directed graph having $|V| = n$ vertices and $|E|$ edges. Additionally, let us consider W be an $n \times n$ weight matrix such that the weight of element $w_{i,j}$ is defined as [3]

$$w_{i,j} = \begin{cases} 0 & \text{if } i = j \\ \text{weight of edge } (i,j) & \text{if } i \neq j \text{ and } (i,j) \in E \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E \end{cases} \quad (1)$$

We then define $d_{i,j}^{(k)}$ be the weight of shortest path from the vertex v_i to the vertex v_j using vertices only from the set 1,2,...,k as intermediate vertex. When $k = 0$, we have $d_{i,j}^{(0)} = w_{i,j}$. After that, we can define a recursive formulation for calculation of all-shortest paths as

$$d_{i,j}^{(k)} = \begin{cases} w_{i,j} & \text{if } k = 0 \\ \min\left(d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\right) & \text{if } k \geq 1 \end{cases} \quad (2)$$

Now, combining Equations 1 and 2 and modifying algorithm Transitive Closure (G) we can solve APSP problem using algorithm Floyd-Warshall (W) [8]:

```
Floyd-Warshall ($W$)
1: $n \leftarrow rows(W)$
2: $D^{(0)} \leftarrow W$
3: for $k \leftarrow 1$ to $n$
4:     for $i \leftarrow 1$ to $n$
5:         for $j \leftarrow 1$ to $n$
6:             $d_{i,j}^{(k)} \leftarrow \min\left(d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\right)$
7: Return $D^{(n)}$
```

The expression $d_{i,j}^{(k)} \leftarrow \min_{k=1}^{n}\left(d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\right)$ is similar to the expression defining the product of two matrices $c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$. In fact, we can use ideas of matrix multiplication presented in CUDA programming manual [6] in parallelization of the Floyd-Warshall algorithm.

4 PARALLEL IMPLEMENTATION USING CUDA

The CUDA programming model extends standard C/C++ with a number of parallel programming abstraction, namely a hierarchy of threads, barrier synchronization, and shared memory. Kernels of a CUDA program can only operate out of device memory that can be allocated either as linear memory or CUDA arrays [6]. Let us consider memory allocation for the matrix presented in Figure 3. It can be done as:

```
int N = 16;
int Size = N * N * sizeof(float);
float d_A;
cudaMalloc(&d_A, Size);
```

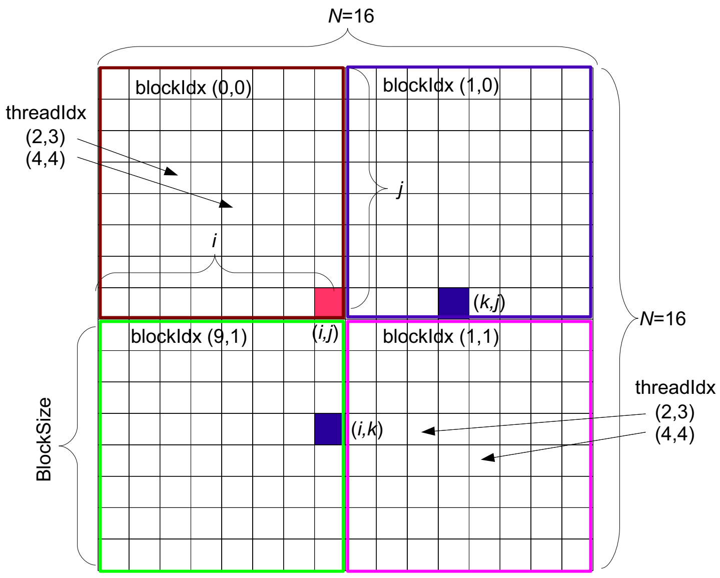


Figure 3: Schema for division in blocks and memory references

Copying of host matrix h_A to device vector d_A can be done as `cudaMemcpy(d_A, h_A, Size, cudaMemcpyHostToDevice);`. As we mentioned in Section 2, the task is divided in grid of block such that each block consists of a number of independent threads and a block is allocated to a multiprocessor. Let us consider the 16×16 matrix presented in Figure 3 and division of the work in four 8×8 blocks. An example of CUDA code to do that is as:

```
BLOCKSIZE = 8;
...
dim3 dimBlock(BLOCKSIZE, BLOCKSIZE);
dim3 dimGrid(N / dimBlock.x, N / dimBlock.y);
MyKernel<<<dimGrid, dimBlock>>>(d_A, N)
```

For now on, the kernel program executes 256 threads divided in four blocks indicated by `blockIdx.x` and `blockIdx.y`. Each block consists of 64 threads indicated by `threadIdx.x` and `threadIdx.y`. During the execution of Floyd-Warshall algorithm, during each execution step of *k* loop, each thread is responsible for the evaluation of its distinctive result value.

Let consider evaluation of value at `d_A[i,j]`. The thread responsible to evaluate the value has `blockIdx.x` = 0, `blockIdx.y` = 0, `threadIdx.x` = 7, `threadIdx.y` = 7 as presented in Figure 3. Additionally, it needs data from `d_A[i,k]` and `d_A[k,j]`. Row and column indices of needed data in devices linear memory can be evaluated as `Row = BlockIdx.y * BLOCKSIZE + threadIdx.y` and `Row = BlockIdx.x * BLOCKSIZE + threadIdx.x`. The core code of kernel can be presented as

```
Row = BlockIdx.y * BLOCKSIZE + threadIdx.y
Col = BlockIdx.x * BLOCKSIZE + threadIdx.x
for (kk = 0; kk < NNODES; kk++)
  if (A[Row][Col] > (A[Row][k] + A[k][Col]))
    A[Row][Col] = (A[Row][k] + A[k][Col])
```

5 EXPERIMENTAL RESULTS

We tested our implementations on AMD Athlon dual core running on 64-bit Fedora 10 Linux distribution. The GPU card used were

NVIDIA GeForce 9600 GT having 64 cuda cores, 512 MB of memory, and running at 720 MHz.

Figure 4 presents the computation times for serial C and CUDA versions of APSP. The graphs indicates that data transfers between CPU and GPU dominate the evaluation time of CUDA implementation when the matrix size is small. The time complexity of serial C version is $O(n^3)$, where n is the size of matrix. The graphs support the argument. According to the graphs, the CUDA implementation offers at least ten-fold speedup in comparison with the serial C implementation.

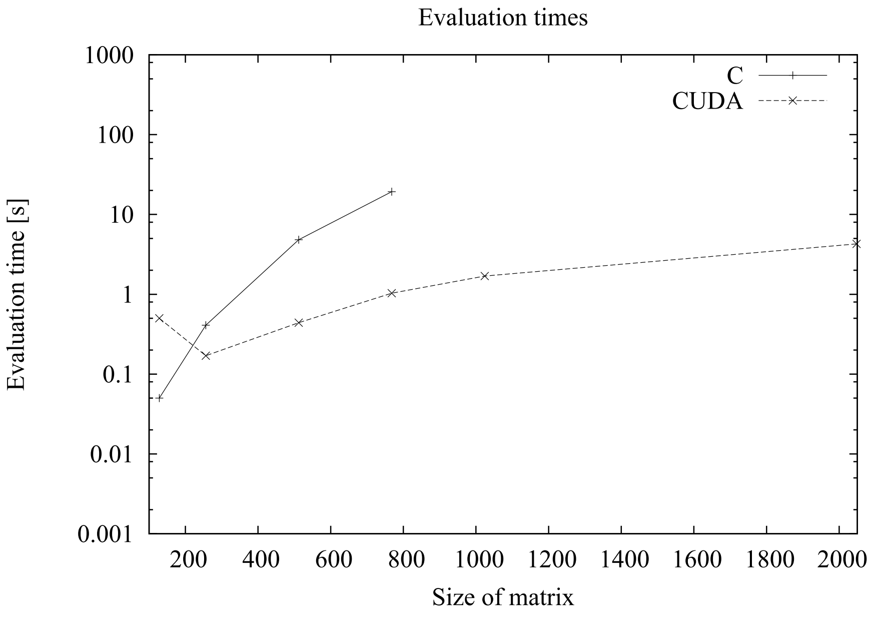


Figure 4: Evaluation times of matrix operations.

The implementations we have presented have a number of drawbacks: Firstly, we didn't take into account sparseness of matrix presenting actual road networks. If this property is taken into account, the evaluation time of serial implementation can be decreased substantially. Secondly, shared memories of multiprocessors were omitted. The use of shared memory should decrease the evaluation time of CUDA implementation. Thirdly, in order to optimize global memory loads and stores we should use 32-, 64-, or 128-bit data types.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we presented calculation of all-pairs shortest path problem based on Floyd-Warshall algorithm and matrix operations. We concluded that the GPU based implementation is superior in comparison to its serial counterpart. However, we did not take into account sparseness of matrices based on real world routing problems.

A continuation to our work is to implement matrix based all-pairs shortest path algorithms on GPU such that the sparseness of matrices has been taken into account. Additionally, the efficient use of shared memory should decrease the evaluation time of CUDA implementations as well.

References

[1] Garland, M.: Sparse matrix computations on manycore GPU's In *Proceedings of the 45th annual Design Automation Conference*. 2008.

[2] Harish, P., Narayanan, P. J.: Accelerating large graph algorithms on the GPU using CUDA. In *proceedings of IEEE International Conference on High Performance Computing* (HiPC). IEEE, 2007.

[3] Katz, G. J., Kider Jr., J. T.: All-Pairs Shortest-Paths for Large Graphs on the GPU. *Graphics Hardware*, 2008.

[4] Nuutila E.: An efficient transitive closure algorithm for cyclic digraphs. *Information Processing Letters* 52, (1994), pp. 207–213.

[5] NVIDIA Corporation: NVIDIA corporation home pages. Available: http://www.nvidia.com/page/home.html. Downloaded Sep 2, 2011.

[6] NVIDIA Corporation: NVIDIA CUDA C Programming Guide. Available:
http://www.nvidia.com/object/cuda_home_new.html.
Downloaded Sep 2, 2011.

[7] Okuyama, T., Ino, F., Hagihara, K.: A Task Parallel Algorith for Computing the Costs of All-Pairs Shortest Paths on the CUCA-compatible GPU. In *Proceedings of 2008 IEEE International Symposium on Parallel and Distributed Processing with Applications*, pp. 284–291. IEEE, 2008.

[8] Cormen, T. H., Leiserson C. E., Rivest R. I., Stein C.: *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

XML and RDF for Semantic Interoperability in Public Administration

Konstantin Hyppönen
University of Eastern Finland,
School of Computing
P.O.B. 1627, FI-70211, Kuopio, Finland
Konstantin.Hypponen@uef.fi

Abstract. In this paper, we discuss the main features and initial design goals of current structured document formats with regard to their applicability for data exchange in public administration. We argue that semantics should not be expressed using element names in document formats. Instead, public administration should switch to more intensive use of semantic technologies and presentation markup.

1 INTRODUCTION

People have used documents for communication, business and administration for years. With the move to computer-based publishing and electronic processes, documents are now commonly stored, processed and archived as computer files. In the history of document standards, three main phases can be identified:

1. Simple text files
2. Binary document formats
3. Structured document formats

First, documents were processed using simple text editors. The switch to binary document formats was mainly driven by the need for better formatting of documents. Structured documents were developed to simplify automated processing of documents or document parts.

In this article we concentrate on the coupling between semantics and document formats. We describe the use of structured documents in public administration and illustrate the need for more precise analysis of semantics conveyed by them. We argue that the current development of structured document formats is driving public administration more towards the use of semantic technologies.

2 SYNTAX, SEMANTICS OR PRESENTATION?

Structured document formats normally describe the rules for constructing text-based documents enriched with markup of certain document parts or characteristics. Well-known examples of structured document formats are HTML and LaTeX. We use these examples here to illustrate some of the features provided by document formats, with the purpose of identifying different markup types:

Syntax The document format specifies a syntax that should be used for document markup. In HTML and XML, characters < and > are used to mark the elements that specify the parts of the document. In LaTeX, a reverse slash \ is used for the same purpose. Other syntax rules may describe the markup of element attributes, mechanisms for inclusion of binary data, or ways to include comments in the source code of the document.

Semantics A document format may include a range of ready-made tags for marking the semantics of the document text. For example in LaTeX, environment *theorem* can be used for signalling that the text of the paragraph describes a theorem. In HTML, the element <title> specifies that the text is the caption of the document, and the element <address> defines the contact details of the documents author. In addition, a document format may specify a way for adding new tags for semantic markup. In LaTeX, new tags or environments may be added by the user in a given document in such a way that the document is still displayable on other systems.

Presentation Often, the document structure includes instructions for systems that display or print the document. The instructions can be direct or implied. An example of a direct instruction is the element <i> in HTML or tag \textit in LaTeX, both of which specify that the text should be rendered in italics. Implied instructions are usually embedded in the structured document processors or renderers. A reference rendering of a heading (<h1> in HTML or \section in LaTeX) might imply the use of the bold font face and a bigger font size. Thus, the document is rendered in a certain way by the processor, although there is no explicit formatting information in the document.

Older structured document formats show rather tight integration of semantics and presentation. Perhaps, this is due to the history of binary document formats. The latest development of structured document formats shows more clear separation of semantics and presentation. This follows the principle of the "separation of concerns", formulated by Dijkstra in 1974 [2]. Presentation features are intended to be described in either a separate document (style sheet), or agreed upon and implemented in a certain way by the rendering engine. The current understanding is that the document itself should contain only semantics, while the presentation features ought to be described elsewhere. The task is not always easy due to historical reasons. For example, the most recent draft of HTML5 retains the element <i>, but defines it in a different way:

"The i element represents a span of text in an alternate voice or mood, or otherwise offset from the normal prose in a manner indicating a different quality of text, such as a taxonomic designation, a technical term, an idiomatic phrase from another language, a thought, or a ship name in Western texts."

One can see that the semantic meaning of this element ("different quality of text") is rather vague. Indeed, the specification even advises authors to check whether other elements are more appropriate:

"Authors are encouraged to consider whether other elements might be more applicable than the i element, for instance the em element for marking up stress emphasis, or the dfn element to mark up the defining instance of a term."

In addition, the note says, 'Style sheets can be used to format i elements, just like any other element can be restyled. Thus, it is not the case that content in i elements will necessarily be italicized." The result is an element with an indefinite semantic meaning and no precise specification of presentation.

3 STRUCTURED DOCUMENTS IN PUBLIC ADMINISTRATION

Public administration is one of the biggest consumers of documents and document standards. Traditionally, public administration has relied on document processors included in office suites. With the recent move to eGovernment and electronic processes, the use of XML-based document formats has increased. XML is a metaformat, which does not define any ready-made tags for semantic markup, but specifies syntax and a model for defining grammars for other document formats. Grammars are expressed in the form of schemas, for example, using the W3C XML Schema specification. Numerous XML-based document formats and XML schemas have been developed for public administration. XML documents or messages are used for exchanging information between computer systems. However, the document contents is often prepared or consumed by humans.

We illustrate the requirements placed on structured document formats for public administration by examining the case of Finnish social services. The following requirements were taken into account:

1. The document should convey precise semantic meanings of data items

2. It should be possible to render or print out the document
3. It should be possible to archive the document in such a way that its content is accessible despite the changes in the semantics of the field.

In order to achieve the first requirement, we started examining the semantics of social care documents. Social services in Finland use about 240 different document types, grouped in 15 categories (applications, decisions, agreements, etc.). Document structures vary greatly. Some document types are highly structured, with small semantic units [5] such as "the coded type of the income of the applicant", while others contain only a few headings, with the rest of contents remaining effectively unstructured. We performed the semantic analysis of the document type contents using the Core Components Technical Specification (CCTS) model [8], resulting in some 150 aggregate core components. Aggregate core components are object classes that represent real life concepts appearing in documents. Examples of such concepts are Human, Organisation, Income, or Financial Situation. Core components can be further specified ("qualified") in the context of a certain document type, resulting in aggregate business information entities with more precise semantics.

The result is a data model consisting of about 1000 elements in aggregate core components. The meaning of the elements can be specified more precisely at the document type level. Furthermore, documents can include elements that are used only in a certain document type. Counting all the elements for which precise semantic meaning has been defined, the data model incorporates a few thousand of data elements.

Elements defined in the data model for a certain field could be transformed to document schemas in several different ways:

1. Define a separate schema for each document type or group of document types
2. Define a single schema that includes all these elements

3. Decrease the precision of semantic meanings, switching to more broad concepts. Define a document format schema based on the broad concepts, and provide a means for including more precise meaning in document instances.

We examine these options with regard to the requirements presented above.

The option 1 results in a number of schemas. In our case this number is up to 240. The requirement A is satisfied, as the schemas include precise semantic meanings of elements. However, in order to render or print the documents (requirement B), the same number of style sheets must be implemented and maintained. In addition, archiving (requirement C) becomes cumbersome, as the single archive might contain documents based on 240 different schemas. In addition, as semantics change from time to time, schemas and corresponding style sheets must be updated, increasing their amount (due to archiving requirements, all schema and style sheet versions must be retained).

The option 2 produces a gigantic schema. It enables documents to convey precise semantic meaning of data, and a single (but large) style sheet is sufficient for rendering and printing out the documents. However, updates in the semantic model influence this schema, which has a negative impact on archiving. Furthermore, the size of the schema influences performance.

The option 3 is similar to the approach that document formats currently provide. They define elements with rather vague semantic meaning. However, the concepts defined in, e.g., HTML, DocBook or similar document formats are not even close to the concepts used in public administration (such as provision of social services). Some fields have, however, used this option. For example, the document standard HL7 CDA R2 used in health care is based on a reference information model (HL7 RIM). The document standard includes a limited number of elements (concepts). More precise semantic meanings of document parts are expressed through the links to external code sets. In addition, the internationally defined document standard can be somewhat extended on the national level by adding new elements to represent missing semantics. However, the resulting semantic model is somewhat fragmented, as part of the semantics is defined in the international standard, part in the national extensions, and part in code sets.

4 STRUCTURED DOCUMENT FORMATS WITH NO SEMANTICS?

The semantics of data used in public administration varies greatly depending on the domain. However, many common concepts can be identified, such as object classes describing a Human, Organisation, Vehicle or an Address. Data dictionaries of commonly used core components for business and public administration have been constructed, such as National Information Exchange Model [4] or Core Components Library [7]. Although such models are usually designed for their eventual mapping to XML Schemas [6, 9], other implementations of document formats are also possible. In Finnish National IT Project for Social Services a decision was made in favour of using XHTML+RDFa, with semantic annotation of XHTML elements by linking them to a data dictionary [3].

We argue that structured document formats designed for public administration should not endeavour to express semantics in the names of the elements. One of the reasons is that semantic models tend to develop and change, negatively influencing the stability of the document format. In addition, since it is commonly not possible to implement all possible precise semantics in a single document schema, designers tend to extend it or start "outsourcing" the semantics to external ontologies or code sets. Extensions and external modules make the document format more complicated, and the semantics of the overall model become fragmented. Even more important problem is that semantics which exist in the document format might be used wrongly (in a different meaning), if an element with appropriate semantics cannot be found in the document format.

Instead, we suggest the following strategy for the modellers of data exchange schemas and structured document formats in public administration:

- A data dictionary should be constructed using a reference model such as CCTS, NIEM or more general ISO 11179 [1].
- The data dictionary should be represented as an ontology (RDF/OWL) for referencing it from the document instances.
- The data dictionary may be used for the construction of XML schemas. XML schemas should not be used for describing document formats. Instead, their use should be restricted to the development of message exchanges (with no requirements on archiving, display rendering or printing).
- The document format should concentrate on presentation markup and enable referencing to external data dictionaries.

To summarize, we suggest the separation of domain semantics and the structured document format. The document format should include only minimal semantics necessary for its operation, such as elements for grouping blocks of text and constructing tables. This suggestion follows the principle of the separation of concerns.

For precise modelling of semantics in data dictionaries, their construction should be supported by terminological work. It should be taken into account, however, that it is challenging to combine a data dictionary with a terminological dictionary in the same repository. Therefore, we suggest keeping them in two separate ontologies with different structures, one of which is designed for data modeling, and the other one for terminological work. The ontologies should be linked in order to specify which concepts in the terminological dictionary are represented by which object classes or data fields in the data dictionary. Also the naming should be consistent in both models.

How does XML compare with semantic technologies in this work? Naturally, in the design and maintenance of semantic models semantic technologies should be utilized. For instance, both data dictionaries and terminological dictionaries can be represented using the RDF data model. On the other hand, XML is a powerful metaformat for the design of data exchange interfaces between computer systems. Although in some cases data could be mediated directly in RDF, we argue that XML will retain its positions due to performance advantage and better tool support. At the same time, we suggest that the modelling of semantics directly as XML schemas should be avoided, as better technologies exist for this work.

5 CONCLUSION

In this paper we argued that the construction of structured document formats with element names based on domain semantics should be avoided in public administration. Instead, we suggest that semantic modeling should be performed separately in tight collaboration with domain experts, knowledge engineers and terminologists. Semantic modeling should not influence the construction of a document format, to avoid unnecessary dependencies between these two scopes of development work.

Acknowledgement This work was supported by the Tikesos project (National Project for IT in Social Services).

References

[1] ISO/IEC 11179, Information Technology – Metadata registries (MDR)

[2] Dijkstra, E.W.: On the role of scientific thought. In: Dijkstra, E.W. (ed.) Selected writings on Computing: A Personal Perspective. pp. 60–66. New York, NY, USA: Springer-Verlag New York, Inc. (1982)

[3] Hyppönen, K., Alonen, M., Korhonen, S., Hotti, V.: XHTML with RDFa as a semantic document format for CCTS modelled documents and its application for social services. In: Garca-Castro, R. (ed.) ESWC 2011 Workshops. LNCS, vol. 7117, pp. 229–240. Springer (2011)

[4] National Information Exchange Model: `https://www.niem.gov`

[5] Nei, S.: Semantic document model to enhance data and knowledge interoperability. In: Devedi, V., Gaevi, D. (eds.) Web 2.0 & Semantic Web, vol. 6, pp. 135–160. Springer US (2009)

[6] NIEM Technical Architecture Committee (NTAC): National information exchange model naming and design rules. Version 1.3 (October 2008), `http://www.niem.gov/pdf/NIEM-NDR-1-3.pdf`

[7] United Nations: Core components library, `http://www.unece.org/cefact/codesfortrade/unccl/ccl_index.html`

[8] United Nations. Centre for Trade Facilitation and Electronic Business: Core components technical specification. Version 3.0 (September 2009), `http://unece.org/cefact/codesfortrade/CCTS/CCTS-Version3.pdf`

[9] United Nations. Centre for Trade Facilitation and Electronic Business: XML naming and design rules technical specification. Version 3.0 (December 2009), `http://unece.org/cefact/xml/UNCEFACT+XML+NDR+V3p0.pdf`

Teaching Performance

Simo Juvaste
School of Computing, University of Eastern Finland,
PO Box 111, 80101 Joensuu, FINLAND,
Simo.Juvaste@uef.fi

Abstract. We discuss why and how performance should be a part of IT teaching and daily work of IT professionals. IT professionals often think of performance, but not necessarily in correct situations. The needed level of performance awareness is trivial in most cases, but without the awareness, there will be scalability failures. In most cases, the motivation for high performance programs is actually efficiency. For deeper performance, the key considerations are (mass) memory usage, algorithms, and parallel execution.

1 INTRODUCTION AND MOTIVATION

Many, if not most, IT developers and students think a lot of performance of computer systems. However, many students tend to look on details or other wrong places when trying to ”optimize” things. Also, in software engineering studies and real life projects, the correctness of the system goes, rightly, in front of performance. As computer hardware performance have improved exponentially last four decades, one could presume that software performance could be neglected in almost all cases. It might be true, but everyone must agree that occasionally (or even often) software we use is slower than we wish. We claim that in many cases the slowness is because some developer just forgot to think performance besides reliability. In some key point of the software something was made, e.g., quadratic instead of linear.

The impressive speed of modern PCs may lead some developers to think that computing resources are almost ”infinite”, and there is no reason to optimize anything beyond ”fast enough” for the user. This view ignores, however, two important (or most important) software markets, namely mobile devices and server software. Modern mobile, battery operated devices have also very powerful processing capabilities, but limited power resources. Modern mobile processors can remain long times idle at very low power consumption, and in case of need for processing power, fire up the circuits (using way more power) at need. After the computation, the processor can go to low power idle mode again. The faster (in term of time/clock cycles) the computation was, the less energy was wasted, and the more time the device remains usable. Further current trends in IT are cloud services and server virtualization. In a cloud computing service, the customer buys server capacity in terms of processing power, memory, and network bandwidth. The more efficient (i.e., faster) the server software is, the more users it can serve with the same amount of resources, or the less resources are needed the serve the same amount of users. These topical issues are the methods to motivate students for performance studies in general.

In most software developments, there are no needs for highly optimized solutions. The common CS folklore (attributed to Dijkstra, Hoare, or Knuth, depending the source) about evils of premature optimization of details is, of course, true. We should teach our students to concentrate on most important aspects of performance. In other words, keep is simple. If the performance plan is simple enough, the developers might be able to follow it.

If we seek for performance instead of efficiency, the current and future technique is parallelization. Parallel processing is (finally) becoming a commodity in form of multi core processors. It is still unclear whether all future developing should be made parallel processing aware or not? Probably not yet. In any case, all developers must remember the possibility of concurrence.

In the remaining of this paper, we briefly discuss what to teach for every CS student about performance. First, we try to give the absolute minimum what every developer must remember in all projects to achieve reasonable efficiency. Second, we discuss the next important issues in achieving high performance.

2 WHAT EVERY DEVELOPER MUST KNOW AND REMEMBER ABOUT PERFORMANCE

When students come from programming courses to data structures and algorithms courses, they already can use arrays and quickly lists as well. In the early exercises in the data structures course, most algorithm performance mistakes made by students are related to wrong usage of arrays and linked lists. Modern programming language libraries, such as JavaSE, C++, C♯, or glib provide convenient implementations of lists and arrays. The convenience masks the underlying data structure, but unfortunately it masks also the performance. It is very easy for the programmer to accidentally use the array as a list, or list as an array. In both cases, some operations suddenly take linear time. Make the mistake in a loop, and we have quadratic time. Quadratic time goes without notice as long as we test the program with small input (up to thousands of elements). The programmer is happy with a working system. But, the quadratic complexity will be a problem later if the component is used for millions of elements! Even worse mistakes might make the time complexity cubic. How to avoid such errors? There is no other way than to keep time complexity in mind in design and implementation. In the application program, this is usually about trivial for anyone who did the introductionary algorithms course. More difficult is to know and especially remember the time complexity of the library operations. Thus, we should emphasize the correct usage of basic data types. Especially, we should make clear difference between lists and arrays even if most APIs try to hide the difference.

Very similar dangers occur in relational databases. A missing index, sorted usage of a hash, or Cartesian join work "instantly" in a small database, or with single user, but case huge scalability problems with real data sets and several concurrent users. Most of time the problems are easily avoided if the developer is aware of the few pitfalls. How the developers remember of the pitfalls. Database programming is more abstract than the rest of applications, thus there are fewer temptations to do the local "cosmetic" optimizations. On the other hand, as the actual execution is not apparent, the developers won't think in time complexity.

Experimental complexity analysis

With the ever increasing usage of libraries and more complex tools, the traditional complexity analysis in becoming impossible. The libraries provide more or less black boxes with documentation telling only the functionality of each method, but little or no information on time complexity. An educated developer probably can guess some complexities based on his/her knowledge on the most common implementations – which is yet another reason for data structures and algorithms course. Ultimately, the software provider needs to make some guarantees of software performance in seconds, transactions/second, concurrent users, etc. To estimate such measures, the developer need both asymptotic complexity class of the system, and either measured performance of the component (or prototype), or an estimate of it. If we use black box libraries, we need to find out also the complexity class of the component experimentally. It is not very difficult, but it is not a common part of computing curricula. Thus, to be able to extrapolate the performance beyond the test set, we need both algorithmic, and practical skills.

3 WHAT ABOUT PERFORMANCE DETAILS AND OPTIMIZATION?

If premature optimization is evil in 97% of cases, how often optimization is needed? During the 40 years since introduction of the rule, compiler technology has improved a lot. Now the compilers can do 99+% of the local optimization needed. For more complex optimizations, there is still some trics to exploit, but half of the developers probably never need it. Most of the rest might need it few times in their career. The rest few cases probably should be left for performance specialists. In most projects, there is no need for a performance specialist, but in some cases there should be a guru designing all the algorithms and profiling all the code.

If the algorithm is efficient, then the next things to consider are memory access patterns. True random access memory is not used anymore. All high performance systems have fairly high memory hierarchy from caches to mass storage. Standard DRAM memory types have high bandwidth relatively high random access times. Thus the keys for efficient memory usage are sequential access and cache usage optimizations. These can easily make a $10\times$ difference in performance. In mobile and virtual machine environments, the difference might be lower, though.

One of the common culprits for system delays have been mass memories, especially rotating hard disks. The disks provide also very high sequential performance, but very poor random performance. Thus, there are good possibilities for disasters and improvements. The recent remedy for mass storage performance problems has been Flash technology. Flash provides clear improvement (up to $100\times$) in random access times, and little edge on bandwidth. Flash, however, has its own performance issues related to the writing mechanism used. It should not used just as a replacement for rotating magnetic disks.

4 ARE WE READY FOR PARALLEL PROCESSING?

Parallel processing has been the key for the best performance since the dawn of computing. Until recently, most of parallel processing has either appeared within the single processor model, or in large scale increasing the system cost considerably. As the clock speeds of processors and instruction level parallelism have reached their limits, we have seen multiprocessing become mainstream. Parallel programming is, however, far from mainstream. Parallel processing is about speed (real time benefits). Most software is truly fast enough if it was made efficient (as we described above). There is no use to parallelize inefficient software. Thus, parallel programming probably should be limited for those few % applications where there is so much to compute that the processing power makes difference. As above, it might be that only (small) part of developers actually needs this in near future. They should, however, be aware of the possibility of being involved with a parallel component.

What is important in parallel programming?

Parallel programming is more complex that sequential programming. Thus the possibilities for inefficient result are ever greater that in sequential development. Often a new student in a parallel computing course comes with a program that has linear inefficiency, i.e., not achieving any speedup! Amdahl's law was probably the first guideline for parallel program design. Even if the law seems trivial and outdated, it probably is still the first and most important one to teach for new parallel computing students. Later the goal should be efficiency. If the goal in parallel processing is speed, there are no processors to waste for inefficient algorithms.

Next to efficient algorithms is again memory usage. Shared memory (or interconnection network) bandwidth is an expensive resource and latency increases as the system grows. Most of parallel program optimization is about optimization of memory access (or other data transfer).

5 CONCLUSIONS

Performance is still important, but in almost all cases, we should **keep it simple**. Complex algorithms, memory optimization, and highly parallel algorithms have their places, but not in everyday development. Everyday development must, however, remember performance next to reliability.

Routing on the OCPC

Anssi Kautonen
Nokia Siemens Networks
P.O.Box 1, FI-02022 Nokia Siemens Networks, Finland
Anssi.Kautonen@nsn.com

Abstract. Optical communication offers huge bandwidth and makes it possible to build communication networks of very high bandwidth and connectivity. We study routing of the h-relations in optical communication pararallel computer under so called OCPC or 1-collision assumption. In an h-relation each processor is the origin and the destination of at most h-messages.

In this paper we study two cases. One where h is smaller than the number of processor. We introduce Penalty and Thinning algorithms.

The another we study is the case where h is much larger than the number of the processors, $h \gg p \log p$. Our algorithm uses total-exchange primitive to route packets. The algorithm attempts to balance the number of packets between origin-destination pairs. The experiments show that algorithm achieves simulation cost which is very close to 1 when h is large compared to the number of processors.

1 INTRODUCTION

We assume the OCPC (*Optical Communication Parallel Computer*) model (also known as *Local Memory PRAM*, *S*PRAM*, and *Optical Crossbar Parallel Computer*). The OCPC model was first introduced by Anderson and Miller [1], and has been studied in [7], [8], [10], [13], [14], [15], [12], [16], [17], [2], [19] and [20].

The memory of OCPC is divided into modules, one module per processor. Communication network is a complete network, thus distance between any pair on nodes is one and the degree of nodes is $p - 1$. Processors communicate with each other by transmitting messages. A processor can transmit a message directly to any other processor and the transmission takes one time unit. At any time unit a processor can send at most one message. The message will succeed in reaching the processor, if it is the only message with that processor as its destination at that time step. If two or more processors attempt to send a message to the same processor, no transmission is successful and a retransmission must occur. This is called the OCPC or 1-collision assumption.

A successful transmission is acknowledged by sending an acknowledgment message/read value to the processor that sent the message. A non-successful transmission is detected by the absence of acknowledgment message. Thus, in constant time all the processors requesting access are informed whether they have succeeded or not. Since each processor can send at most one message and receive at most one message during message sending phase, all acknowledgements are successful in the acknowledgement sending phase.

In this paper we consider balanced communication patterns, called h-relations. Let p be the number of processors in a parallel computer. Let $K = (k_{ij})$ be a $p \times p$ matrix, where k_{ij} gives the number of messages originating at processor i and destined for processor j. If we let h be the maximum sum of any row or column of this matrix; then the matrix specifies an *h-relation*. The problem of solving this communication task is termed the *h-relation problem*.

The problem is motivated by implementation of the shared memory abstraction, for example the PRAM model [5] and the BSP model [20], and also by direct implementation of specific parallel algorithms. The value of h affects the latency parameter of the BSP model, and the effiency of the implementation of the h-relation affects the bandwidth parameter of the BSP model.

If suitable techniques, such as the *slackness* principle and randomized hashing are used, implementation of shared memory can be reduced to efficient routing of an h-relation in a complete network under OCPC assumption. An m processor algorithm, when implemented on an n processor machine, where $n \leq m$, is said to have parallel slackness factor of m/n for that machine. A physical processor simulates virtual processors in a round-robin manner. This helps, under certain conditions, to decrease the amortized cost of packet routing to a small constant per packet. When h packets can be transmitted in $O(h)$ time we say that routing is work optimal.

1.1 Previous results

Anderson and Miller observed that if all processors have complete information about a given h-relation, there exists a schedule which routes the h-relation using exactly h communication steps. This observation is based on Hall's famous theorem on the existence of perfect matchings in bipartite graphs.

We assume that each processor initially only knows about the messages that it wants to send, and it learns about the rest of the h-relation only through the successes and failures of the packets that it tries to deliver. Also, we assume that each processor knows the value of h, when the routing begins.

In *direct* algorithms, the processors send messages directly to their final destination without any intermediate destinations. Goldberg & al. [8] proved that for any (randomized) direct algorithm there is a 2-relation that takes $\Omega(\log p)$ steps to route. MacKenzie, Plaxton and Rajaraman generalized this result by showing for any (randomized) direct algorithm and any $h \geq 2$, there is an h-relation that takes $\Omega(h + \log h \log p)$ expected steps to route. On the other hand, there exists a randomized direct algorithm (Geréb-Graus and Tsantilas [7]) that takes $O(h + \log h \log p)$ routing steps with high probability – so the lower bound is tight.

Even for indirect algorithms it is not possible to achieve time bounds of the form $O(h)$ for small values of h. Golberg et al. [9] proved a lower bound of $\Omega(h + \sqrt{\log \log p})$ for realizing h-relation on the OCPC.

The simplest attempt to implement an h-relation is perhaps the randomized algorithm called the *greedy algorithm* (see Algorithm 1).

By greedy principle, each processor makes a packet transmission attempt at every time step until all packets are transmitted. The fatal drawback of the greedy algorithm is *livelock* situation: Some packets can cause mutual failure of sending until eternity. Consider the situation, when two processors each have one packet targeted to the same processor. Due to greediness they are forced to send — and fail for ever since then. Unfortunately, this situation is very common: At the end of the routing process (unless special care is taken), there are usually several processors, which all attempt to send their message(s) to the same processor(s).

Algorithm 1 Greedy routing algorithm.

```
1: proc Greedy
2: for all processors par do
3:   while processor has packets remain do
4:     choose an usent packet at random try to send ot to its destination
```

The livelock situation can be avoided. Such an algorithm was provided by Anderson and Miller [1], and it was improved by Valiant [20]. These (non-direct) algorithms route work-optimally an h-relation for $h \in \Omega(\log p)$, where p is the number of processors. Other algorithms with even lower latency were proposed by [3, 4, 8, 10, 11] (the problem setting is not exactly the same in all papers). The asymptotically best algorithm is Goldberg et al.'s algorithm [8], which routes an arbitrary h-relation on a p-processor OCPC with $\Theta(h + \log\log p)$ communication steps. If $h \leq \log p$, then the failure probabilty can be made as small as $p^{-\alpha}$ for any positive constant α.

Contrary to these theoretically strong algorithms, the algorithm of Geréb-Graus and Tsantilas [7] has the advantage of being *direct*, i.e. the packets are sent to their targets directly, without intermediate nodes. The algorithm of Geréb-Graus and Tsantilas routes work-optimally h-relations, for $h \in \Omega(\log p \log\log p)$. Indeed, for $h \in \Omega(\log p \log\log p)$, the GGT algorithm routes any h-relation in time $O(h)$ with probability higher than $1 - 1/p^{\alpha}$ for any $\alpha \geq 1$. Also Kautonen & al. have developed direct work optimal algorithms, [14], [15] and [16]. Those algorithms route packets in time $O(h + \log p \log\log p)$, and are thus work optimal when $h \in \Omega(\log p \log\log p)$.

In 1-optimal protocols the focus is on the leading constant factors of asymptotically efficient algorithms, since using a single *total-exchange* operation efficiently requires that $h \in \Omega(p)$. Gerbessiotis and Valiant [6] have presented a 1-optimal algorithm that routes a random h-relation using at most $\frac{h}{p}(1 + o(1)) + O(\log\log p))$ total-exchange rounds with high probability. The total-exchange (also known as *all-to-all personalized communication*) in the OCPC is realized as follows (see Algorithm 2). Protocol runs in $p - 1$ phases, and during phase j processor i transmits its message destined for processor $(i + j)$ mod p. If m is the maximum number of messages over all sender-destination pairs, then m total-exchange rounds are required to route messages.

Rao et al. [19] have presented a better protocol that uses only $\frac{h}{p}(1 + o(1)) + O(\log^* p)$ total-exchange rounds with high probability. In these results, the leading constant factors are very small, i.e. clearly less than e when $h \gg p$.

2 PENALTY ALGORITHM

In the Penalty algorithm (see Algorithm 3) the control of the transmission rate is based the number of unsuccessful transmission attempts the packets have had. The packet transmission probability is a function f of the the number of unsuccessful transmission attempts that the packet has had. If this function f is monotonically increasing, then the Penalty algorithm is livelock free. For example f can be logarithm, square root, linear or square function. When packets targeted to processor p collide, the value of function f increases for those collided packets. Thus, transmission probability of those packets decreases. As the number of collisions of the packets targeted to processor p increases, the expected number of packets tried to transmit ot processor p during one time step decreases be-

Algorithm 2 Total-exchange algorithm

```
1:  proc Simple total-exhange($p$, $h$)
2:  for all processor $P_i$ par do
3:      $t = (P_i + 1) \bmod p$
4:      while packets remain do
5:          if  processor has a packet to send to processor $t$ then
6:              Send that packet to processor $t$
            $t = t + 1$
7:          if $t = P_i$ then
8:              $t = t + 1$
9:          if $t = p$ then
10:             if $P_i = 0$ then
11:                 $t = 1$
12:             else
13:                 $t = 0$
```

low 1. Hence, the packet transmission will be eventually successful.

Algorithm 3 Penalty algorithm

```
1:  proc Penalty($f$:function)
2:  for all processors $P$ par do
3:      while processor $P$ has sent packets do
4:          choose a packet $x$ at random
5:          if  1/(number of failures of $x$)  $\geq$  RandomNumber[0..1)
            then
6:              attempt to send $x$
```

The idea used in this algorithm is similar to used in the Ethernet. There are however several differences between the Penalty algorithm and the Ethernet algorithm. In the penalty algorithm each packet has its own counter, which is augmented by one by one when a transmission fails du collision. The packet that is attempted to transmit is chosen uniformly at random instead of the head of queue as it is done in the Ethernet. Messages in the Penalty algorithm are never discarded.

3 THINNING ALGORITHM

In basic form of the Thinning algorithm (see Algorithm 4), the packets are routed phase by phase. In each phase, each packet (in random order) is tried at most once. The expected number of transmission attempts per time unit has a fixed value throughout the phase. Thinning by factor t (where $t > 1$) means making h packet sending attempts in $t \times h$ units of time. (A similar effect could be achieved by transmitting with probability $1/t$.) Thinning factor may or may not change from phase to phase.

This algorithm is inspired by Gerb-Graus and Tsantilas algorithm [7], and Paterson and Srinivasan's algorihm [18].

Algorithm 4 Thinning algorithm

```
1: proc Thinning($h$,$h_0$,$t$,$\tau$)
2: for all processor $P$ par do
3:     $i := 1$
4:     while $h > h_0$ do
5:         Try to transmit $h$ packets (if so many remain) in the following $\lceil t(i)h \rceil$ steps
6:         $h := (1 - e^{-1/\tau(i)})h;\ i := i + 1$
7:     while packets remain do
8:         Try to transmit the remaining packets in the following $\lceil t(i)h_0 \rceil$ steps
```

For each time step of the i'th phase, there are the same expected number of packet transmissions. In the first phase each processor has at most h packets, but as the algorithm moves to the next phase (next round of first while-loop), the value of h represents only a certain kind upper bound for the expected degree of the current h-relation. Especially, this means that a processor may have more than h packets when a phase begins, but only tries to send h of those.

The larger value $t(i)$ is the higher the probability of successful transmissions is. However, the drawback is that a processor can not successfully transmit a packet at those moments when it does not even try to send. Thinning by factor $t(i)$ would thus imply inefficiency by factor $t(i)$. However, the increasing packet transmission probability compensates this disadvantage.

The function $t(i)$ is related to the function $\tau(i)$ that charecterizes how the problem size h_i in beginning of the $i'th$ phase is quanranteed to decrease to h_{i+1} at the end of the phase. The expectd number of successful transmssions per processor during a phase i is $h_i/e^{1/t(i)}$. The function $\tau(i)$ must selected so that there remaind an h_{i+1}-relation with high probability after phase.

Even though transmission probabilyt less than 1 elinates the livelock, it does not guarantee steady throughput. When the degree of h-relation decreases, the packet transmission attempts become less and less random. There is a high risk that number of packets per processor becomes unbalanced. If the lenght of routing phase is close to 1 there is a danger of repeated collisions. That is why there is a lower limit $h_0 \in \Omega(\log p)$ for the number of time steps per phase. The level h_0 will achieved after $O(\log(h/h_h0))$ phases, if the degree of the h-relation can be decreased during each phase by at least some fixed constant factor.

Theorem in [16] state following:

Theorem 3.1 *Let $h_0 = \Theta(\log p)$, and t and τ be the functions in the Thinning algorithm. If $1 \leq \tau(i) \leq \alpha t(i)$ and $t(i+1)/t(i) \leq \beta\tau(i)$ for some constants $0 < \alpha < 1$ and $0 < \beta < 1$, and $h \in \Omega(\log p \log\log p)$, then the Thinning algorithm routes any h-relation in time $O(h)$ with high probability.*

The conditions for thinning factor t and compression factor τ are quite general – the most essential requirement is that $t(i) \geq \tau(i)$. Consider the following four cases.

Case I: Consider situation, where $1 < t(1) = t(2) = t(3) = \ldots = d$, $1 \leq \beta'\tau(i) = \alpha' d$, $0 < \alpha', \beta' < 1$, and for c it holds that $1 - e^{-1/\tau(i)} = c$. This is the case of *constant* thinning (CT) [14], and it satisfies the conditions of Theorem 3.1.

Case II: $t(i) = 1 + i \times d$ for some $d > 0$. For a properly chosen d, it is possible to define $\tau(i)$ so that it satisfies the conditions of Theorem 3.1. This is called *linear* thinning (LT) in [14].

Case III: $t(i) = \delta d^{i-1}$ and $\tau(i) = d^{i-1}$ for some $\delta, d > 1$. This case is called *geometric* thinning (GT) in [14,15]. As such it does not satisfy the conditions of Theorem 3.1, but it can be made to satisfy the conditions by choosing $\tau(i) = d' + d^{i-1}$ for some constant $d' > d$.

Case IV: Let

$$t(i) = t_1 + \frac{(t_2 - t_1)(\tanh(i-k) - tanh(-k))}{1 - \tanh(-k)},$$

for some $1 < t_1 < t_2$, $k \geq 1$. This is a *sigmoid* thinning (ST) algorithm. Also in this case it is possible to define $\tau(i)$ so that the conditions of Theorem 3.1 are met.

The analysis of the algorithm can be found in [16].

4 BALANCE ALGORITHM

The Penalty algorithm and Thinning algorithms were developed case were $h < p$. We have also developed algorithm for case where $h > p$ (see Algorithm 5). The algorithm is collision-free, no special acknowledgement step is required. The algorithm is based on total-exchange and it is indirect, thus some packets are sent to final destination via an intermediate destination(s).

In order to be efficient, total-exchange based algorithms require a large h/p-ratio. The problems in a simple total-exchange scheme are that the number of the messages between origin-destination pairs in not balanced, and routing two messages from processor P_i to processor P_j takes at least p routing steps.

The expected number of packets over any any origin-destination pair is h/p. However the number packets over all origin-destination pairs is not uniform. Some pairs have "excess" of packets and some pairs have "defiency" of packets with respect to the average value.

The standard deviation of the number of packets over all origin-destination pairs is $\Theta(\sqrt{h/p})$, thus to route all packets to their final destinations would require at least $h/p + \Omega(\sqrt{p/h})$ total-exchange rounds using the Algorithm 2. Especially the routing of the last packets takes many routing steps. The maximum number of messages between any origin-destination pair determines the number of routing steps required to route all messages to their final destinations.

We have developed an algorithm that balances the load so that the number of messages over all origin-destination is approximately equal. Routing is done using the total-exchange protocol. If the processor has no packet to send to current target processor, then it checks to which processor it has most packets to send. Let m be the maximum number of messages to any other processors. Let $x_1, \ldots, x_n$ be a set of processors to which the current processors have m packets to send. If m is greater than one, then one of packets destined to one of processors $x_1, \ldots, x_n$ is chosen uniformly at random. If m is one, then the last packets to some destination are not sent because a packet may travel from intermediate destination to intermediate long before the packet arrives its final destination.

Assume that $h >> p \log p$, then routing random h-relation then routing a random h-relation takes $\frac{h}{p}(1 + o(1)) + O(\sqrt{\frac{h}{p}\log p})$ total-exchange rounds. Arbitrary h-relations can be routed about in twice that time, by routing messages first to random intermediate destinations and then to their final destinations as proposed by Valiant [20]. The whole analysis can be seen [12].

5 EXPERIMENTAL RESULTS

We have developed a routing simulator. It can simulate several kinds of routing algorithms for complete optical networks. The result of simulation is the average routing *cost*. The cost is the number of steps required to route all packets, divided by h. In the experiments packets are routed to final destinations. The minimum simulation cost is 1. A processor needs at least 1 step per packet. The

Algorithm 5 Balance algorithm

```
1:  **proc** Balance ($p$, $h$)
2:  **for** all processors $P_i$ **par do**
3:      $t = (P_i + 1) \bmod p$
4:      **while** packets left **do**
5:          **if** processor has a packet to send to processor $nt$ **then**
6:              Send that packet to processor $nt$
7:          **else**
8:              Let $m$ be the maximum number of packets left that the
                current
9:              processor has to send to any other processor
10:             **if** $m > 1$ **then**
11:                 Let $x_1, \ldots, x_n$ be the set of processors for which
12:                 the current processor has $m$ packets to send
13:                 Select uniformly at random one of packets targetted
14:                 to some of processors $x_1, \ldots, x_n$.
15:                 Send the selected packet to processor $nt$
            $t = t + 1$
16:         **if** $t = P_i$ **then**
17:             $t = t + 1$
18:         **if** $t = p$ **then**
19:             **if** $P_i = 0$ **then**
20:                 $t = 1$
21:             **else**
22:                 $t = 0$
```

results are averages of 1000 experiments.

5.1 Assumptions

We make the following assumptions.

- The machine has p processors. Each processor one optical transmitter and one optical receiver. During a time step a processor can send and receive at most one packet.
- It is assumed that processors have a common clock. All processors accomplish the packet transmission step at the same time.
- In each experiment, each processor has h packets to send. The destination of packet cannot be the sending processor itself.
- Simulation is stopped after all packets are sent, or if there is a livelock. We classify as as livelock situation, where the simulation cost becomes higher than 100. However, in the algorithms we have studied in this paper, the real livelock situation is impossible, and simulation cost over 100 extremely unlikely.

When investigating the results of the Penalty algorithm, it seems that when p is small or when h is small compared to p compared to p then logarithmic and square root functions are good choces as a backoff function f. For large p linear function is the best choice.

Thinning algorithms require more experiments in order to find good values for parameters. Different thinning methods give quite similar results.

When comparing the Penalty algorithm and the Thinning algorithms it seems when h is small compared to p the Penalty algorithm is the best choice, but when $h > \log p$ Thinning algorithms are the best choice. When $p = 1024$ and $h = 2$ then the simulation cost is with the Penalty algorithm is 8.0 and with the Geometric Thinning algorithm 10.8. And when $p = 1024$ and $h = 1024$ the simulation cost is 3.5 with the Penalty algorithm 3.0.

When $h > p$ the Balance algorithm is the better option than the Penalty algorithm or the Thinning algorithm. When h is very large compared to p, the best result achieved using the Penalty algorithm or the Thinning algorithm is $e \approx 2.7$. However with the Balance algorithm the simulation cost approaches 1 as the value of h increases. When $p = 1024$ and $h = 1024$ the simulation cost with Balance algorithm is 2.0 and when $p = 1024$ and $h = 32 * p$ then the simuation cost is 1.1.

More simulation results available in [13], [14], [15], [12], and [16].

References

[1] R. J. Anderson and G. L. Miller. Optical Communication for Pointer Based Algorithms. Technical Report CRI-88-14, Computer Science Department, University of Southern California, LA, 1988.

[2] F. Meyer auf der Heide, K. Schröder, and F. Schwarze. Routing on networks of optical crossbars. *Theoretical Computer Science*, 196(1–2):181–200, 1998.

[3] A. Czumaj, F. Meyer auf der Heide, and V. Stemann. Shared Memory Simulations with Triple-Logarithmic Delay. *Lecture Notes in Computer Science*, 979:46–59, 1995.

[4] M. Dietzfelbinger and F. Meyer auf def Heide. Simple, Efficient Shared Memory Simulations. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 110–119, Velen, Germany, June 30–July 2, 1993. SIGACT and SIGARCH. Extended abstract.

[5] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proceedngs of the 10th ACM Symposium on Theory of Computing*, pages 114–118, 1978.

[6] A.V. Gerbessiotis and L.G. Valiant. Direct Bulk-Synchronous Parallel Algorithms. *Journal of Parallel and Distributed Computing*, 22(2):251–267, 1995.

[7] M. Geréb-Graus and T. Tsantilas. Efficient Optical Communication in Parallel Computers. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 41–48, San Diego, California, June 29–July 1, 1992. SIGACT/SIGARCH.

[8] L.A. Goldberg, M. Jerrum, T. Leighton, and S. Rao. A Doubly Logarithmic Communication Algorithm for the Completely

Connected Optical Communication Parallel Computer. *SIAM Journal on Computing*, 26(4):1100–1119, 1997.

[9] L.A. Goldberg, M. Jerrum, and P.D. MacKenzie. An $\Omega(h + \sqrt{\log\log n})$ Lower Bound for Routing in Optical Networks. *SIAM Journal on Computing*, 27(4):1083–1098, 1998.

[10] L.A. Goldberg, Y. Matias, and S. Rao. An Optical Simulation of Shared Memory. *SIAM Journal on Computing*, 28(5):1829–1847, 1999.

[11] R. K. Karp, M. Luby, and F. Meyer auf der Heide. Efficient PRAM Simulation on a Distributed Memory Machine. *Algorithmica*, 16(4/5):517–542, October/November 1996.

[12] A. Kautonen. Fast Total-Exchange Algorithm. *Lecture Notes in Computer Science*, 3758:524–529, 2005.

[13] A. Kautonen, V. Leppänen, and M. Penttonen. Simulations of PRAM on Complete Optical Networks. *Lecture Notes in Computer Science*, 1124:307–310, 1996.

[14] A. Kautonen, V. Leppänen, and M. Penttonen. Constant Thinning Protocol for Routing h-Relations in Complete Networks. *Lecture Notes in Computer Science*, 1470:993–998, 1998.

[15] A. Kautonen, V. Leppänen, and M. Penttonen. Thinning Protocols for Routing h-Relations in Complete Networks. In *Proc. of International Workshop on Randomized Algorithms*, pages 61–69, Brno, 1998.

[16] A. Kautonen, V. Leppänen, and M. Penttonen. Thinning Protocols for Routing h-Relations Over Shared Media. *Journal of Parallel and Distributed Computing*, 70(8):783–789, 2010.

[17] P.D. MacKenzie and V. Ramachandran. ERCW PRAMs and Optical Communication. *Theoretical Computer Science*, 196(1–2):153–180, 1998.

[18] M.S. Paterson and A. Srinivasan. Contention Resolution with Bounded Delay. In *Proceedings of 38th Annual IEEE Symposium on Foundations of Computer Science*, pages 104–113, 1995.

[19] S.Rao, T. Suel, T. Tsantilas, and M. Goudreau. Efficient Communication Using Total-Exchange. In *Proceedings of 9th IEEE International Parallel Processing Symposium*, pages 544–555, 1995.

[20] L.G. Valiant. General Purpose Parallel Architectures. In *Handbook of Theoretical Computer Science, Ed. Jan van Leeuwen, Elsevier and MIT Press*, volume 1. Elsevier Science, 1990.

Learning and Teaching Computer Science

Marja Kuittinen
University of Eastern Finland
School of Computing
P.O. Box 111, FI-80101 Joensuu, Finland
marja.kuittinen@uef.fi
http://www.uef.fi/cs

Abstract. This paper discusses the practice of learning and teaching computer science (CS). The first part includes an overview of the ACM curricula guidelines and other learning objectives, a short description of learning paradigms, and a characterization of CS students as learners. The second part consists of a description of the body of knowledge that CS teachers should master including issues such as the history of the discipline and the role of computers in education. This is followed by some options for teaching, that is, hardware and software suitable for teaching CS. Finally, there is a discussion of the role of CS education research emphasizing that it is essential for CS teachers to follow up the relevant literature in order to keep up-to-date.

1 INTRODUCTION

Computer science education is given practically worldwide. Education is available in various schooling levels from K-12 (primary and secondary education) to universities. The reason for educating computer scientists is obvious: there is a need for proficient computer science (CS) professionals who are able to produce quality products—software and hardware. In this paper we focus on CS education given at the university level. We are interested in learning and teaching CS, and stress the role of CS education research as a basis for quality education.

Computing has been taught from the late 1950's. Since then, CS educators and researchers have been trying to find out what are the essential topics of computing, what topics are recommendable, and how to teach them in an efficient way. At the moment, thanks to the work of the Association for Computing Machinery (ACM), the computing community has detailed curriculum guidelines for both undergraduate and graduate programs of computing. Today there are more than 300 universities around the world offering degree education in computer science (TopUniversities, 2011).

Over decades the general belief of human learning has shifted from pure behaviorism to cognitivism, having impact on CS education, also. A lot of multidisciplinary research has been reported, but there are still many open questions, such as how to teach programming fruitfully, or how to concretize abstract issues that are so common in computing.

The rest of this paper is divided into two parts: learning computer science and teaching computer science. The first part presents what undergraduate CS students should learn of the discipline, followed by a short review of learning paradigms and a description of CS students as learners. The second part consists of the body of material that CS teachers should master, some options for teaching CS, and, finally, an introduction to CS education research with a short list of top publications, that all teachers should have their eyes on in order to keep up-to-date in education.

2 LEARNING COMPUTER SCIENCE

This section describes briefly what an undergraduate student in CS should at least learn. In addition, there is a short review on learning paradigms and theories.

2.1 What should be learned in CS?

The Association for Computing Machinery (ACM) has been tailoring curriculum recommendations since 1960's (ACM, 2011a). The first one (Conte et al., 1965) presented 16 courses divided into *required* (containing 5 courses), *highly recommended electives* (4), and *other electives* (7). These 16 courses concerned topics such as algorithmic processes, programming, compilers, information structures, numerical calculus, and logic design.

Since then, the ACM has endeavored, together with leading professional and scientific computing societies, to tailor curriculum recommendations to the computing at approximately ten-year intervals. The latest curricula recommendation—Computer Science Curriculum 2008 (aka CS2008; see ACM, 2008)—is an interim review of the Computing Curricula 2001 Computer Science (aka CC2001; see ACM, 2001) volume, whose Body of Knowledge has been updated by the guidance of consultations and discussions with several quarters (industry, academia, etc). Originally, the CC2001 was based on the CC1991 which was strongly influenced by Denning & al. (1989). In CC2001, the term *core* (used for the first time already in 1978 in this contex) was defined. The definition says:

> "The core is the set of units for which there is a broad consensus that the material is essential to an undergraduate degree in computer science."

The CS2008 Body of Knowledge has 14 topics with 280 core hours; one core hour being equivalent to 60 minutes. The 14 topics are as follows: Discrete Structures (43 core hours), Programming Fundamentals (47 core hours), Algorithms and Complexity (31 core hours), Architecture and Organization (36 core hours), Operating Systems (18 core hours), Net-Centric Computing (15 core hours), Programming Languages (21 core hours), Human-Computer Interaction (8 core hours), Graphics and Visual Computing (3 core hours), Intelligent Systems (10 core hours), Information Management (11 core hours), Social and Professional Issues (16 core hours), Software Engineering (31 core hours), and Computational Science (no core hours). The detailed descriptions of the topics can be found in CS2008 Curriculum Update (ACM, 2008).

The Computing Curricula 2005 (aka CC2005; see ACM, 2005) provides undergraduate curriculum guidelines for five defined subdisciplines of computing: *Computer Science, Computer Engineering,* *Information Systems, Information Technology,* and *Software Engineering*. Most of the sub-disciplines have got either a new or updated curriculum since then (see ACM, 2011a).

The core hours are the minimum that an undergradute student should learn in order to get an essential base on which other courses may build. In addition to the ACM's recommendations, there are other things that CS undergraduate students should learn. Boyle & Clark (2004) state that

> "Two of the many things the educated computer scientist might know about are computer science pantheon (idols) and the metaphorical structure of the discipline's technical language."

Their first claim is, that it is necessary for each generation to have some knowledge of the past and to understand how things came to be as they are in order to be able to contribute to the evolution of discipline. Boyle & Clark divide the idols of computing into *eponymy* and *pantheon*: a person whose name is attached to an invention, occasion, or place; and temple of all gods, respectively. They claim that eponymy, as used and recognised in general public, is scarce in the CS. Probably only Turing would be known to the public, thanks to the Turing Test.

According to Boyle & Clark (2004), the existence of the pantheon is a badge of disciplinary maturity, and, thus, a knowledge of it is part of the disciplinary education. In order to find out the pantheon of CS, Boyle (2003) conducted a survey asking higher education teachers

> "To which 8 people in the discipline do you consider all Computing (and similar) graduates should be able to attach a two-sentence biography?"

The top ten is summarised in Table 1 including the views of UK and North American respondents. The detailed results are presented in Boyle (2003).

Table 1: Percentage of poll of the top 10 choices; overall, UK and North American respondents (Boyle & Clark, 2004).

	Overall	UK	North America
Turing	20	20	21
Von Neumann	15,5	13	19
Knuth	12,5	11,5	13
Dijkstra	11,5	7,5	13
Babbage	10	12,5	7
Hopper	7,5	7,5	9
Gates	7	8	5
Wirth	6	6	5
Berners-Lee	6	8,5	5
Lovelace	4,5	4,5	4

Secondly, Boyle & Clark (2004) argue that it is also important to develop understanding of the metaphorical structure of the language used in a discipline. This means that in the case of computing, it is necessary to understand that technical terms are not only a collection of nouns with abstract definitions. For example, the term *network* is itself metaphorical and references a communications network such as a local goods transportation system using the same concepts, e.g., packet, route, address, bridge, collide, etc. Thus, the language of networking is the language of transport systems.

2.2 On Learning and CS

Dozens of learning theories have been created during the history of educational sciences. These theories can be classified in several ways into different paradigms. One of the classifications presents four paradigms (Learning-theories.com, 2011): behaviorism, cognitivism, constructivism, and humanism.

The first paradigm, *behaviorism*, is based on stimulus—response principle. The idea is that all behavior can be explained without the need to consider internal mental states or consciousness. A learner is viewed as a passive object, responding to environmental stimuli. Example theories of this paradigm are Classical Conditioning (Pavlov, 1927) and the GOMS Model (Card, Moran, and Newell, 1983). This paradigm is nowadays obsolete.

Cognitivism emphasises that mental functions can be understood. The learner is an information processor, whose actions are consequences of thinking. Example theories are Cognitive Load Theory (Sweller, 1988), Mental Models (Johnson-Laird, 1983), and the Stage Theory of Cognitive Development (Piaget, 1928). Cognitivism replaced behaviorism in the 1960's as the dominant paradigm.

Constructivism considers learning as an active, constructive process. A learner is an information constructor who actively creates his own subjective representation of the objective reality. Example theories are Communities of Practice (Lave and Wenger, 1998), Discovery Learning (Bruner, 1967), and Situated Learning (Lave, 1988).

Finally, *humanism* sees learning as a personal act. A learner has affective and cognitive needs, that need to be fulfilled. Learning is student-centered and the teacher is a facilitor. Example theories are Experiential Learning (Kolb, 1984) and Maslow's Hierarchy of Needs (Maslow, 1943).

What kind of learners are CS students, then? According to Alaoutinen & Smolander (2010), CS students have some significant differences in their learning styles compared to engineering students and that cultural features, gender, and personal background affect learning. Alaoutinen & Smolander state that most engineering students are active, sensing, visual, and sequential (in terms of Felder Silverman Learning Style Model), whereas CS students are less visual and less sequential when compared to engineering students. These differences should be taken into account when planning teaching.

Another study by Renwick & Foltz (2011) concerned the learning styles of information technology students. They made a short survey of recent research into learning styles, which revealed that IT students prefer kinesthetic learning styles as opposed to visual, aural, and read/write styles. Kinesthetic learners learn best from doing: they use trial and error, complete laboratory exercises, and work problems out in a hands-on manner. Thus, IT students benefit from active learning experiences.

Cukierman & McGee Thompson (2009) describe how they have developed the Academic Enhancement Project (AEP, Simon Fraser University), which has been created to support student learning by integrating activities that introduce students to basic learning theory and strategies into core first-year CS courses. Their findings concerning the usefulness of those activities indicate that such practices are promising to address students' academic challenges, possibly having a positive impact on retention.

3 TEACHING COMPUTER SCIENCE

In this section we first discuss what are the demands for CS teachers. Then we have a few words about the existing options for teaching and end with a short review of the CS education research.

3.1 What should CS teachers know?

It is obvious that CS teachers need to have the mastery of core CS material discussed previously in this paper. In addition, there are other things a CS teacher should be familiar with. Gal-Ezer & Harel (1998) have presented a valuable body of material with relevant bibliography that good CS teachers should master. Such knowledge expands teachers' perspectives on the field enhancing the quality of teaching. The following is a brief summary of the issues raised by Gal-Ezer & Harel.

History of a science provides a global perspective of the field and its structure as well as clarifies its relationship with other fields. Knowing the history of one's discipline generates appreciation to the pioneers of the field and provides a deeper understanding of everything what has been done so far. In addition, the cognizance of the history of the discipline is essential for being able to teach students to know the idols—as Boyle & Clark (2004) suggested.

What is CS? It is useful for a CS teacher to know the unique nature of CS, with its special algorithmic way of thinking and extremely short history. The ACM curricula recommendations give an idea of what the field is really about. In addition to the bibliography proposed by Gal-Ezer & Harel (1998), it might be interesting to read the article by Tedre (2011).

A bird's-eye view of the discipline is essential for CS teachers. Gal-Ezer & Harel (1998) note that "it should be as comprehensive as possible, with depth and detail begin sacrificed for scope and perspective".

Computers in education deals with various approaches to teaching CS. Actually, there exists three totally different directions: disseminating computer literacy; using computers in teaching other subjects; and teaching CS. Realizing this partition is very important for CS teachers, in order to avoid confusing the spirit and the methods of one direction with those of the others.

The problematics of learning and teaching programming includes several difficult questions such as when, to whom, how, and why—indeed, whether—to teach programming. As Gal-Ezer & Harel (1989) state, this topic is not only broad, multisided, and highly controversial, but also crucial in its long-lasting influence on students.

Tools and methods for teaching are essential to know for CS teachers. In the late 1990's the situation was not that good, but nowadays there is an extensive variety of computer-based teaching aids available. A problem is how to find information about them. Using portals, such as the ACM Digital Library, produces a nice list of scientific articles, but presumably not all of the tools and methods can be found that way. There exists repositories for CS, such as Stanford CS Education Library (2006) and CITIDEL Repository (2011), that are discussed in more detail in the next section.

In order to keep up to date with the development of CS education, it is necessary for teachers to read relevant professional periodicals. Suggestions for suitable reading can be found in Gal-Ezer & Harel (1998) and later on in this paper.

3.2 Options for teaching CS

Options for teaching are numerous in any field including computer science. There is an abundance of technical equipment as well as software available for use to support learning.

Educational technology covers both tools and methods used for teaching in general. It relies on a broad definition of the word "technology", which refers to material objects of use to humanity, such as machines or hardware, but can also encompass broader themes, including systems, methods of organization, and techniques (Educational Technology, 2011). Furthemore, educational technology includes, but is not limited to, software, hardware, as well as Internet applications, such as wikis and blogs, and activities.

The Educational technology page in Wikipedia presents a good list of technology available (Educational Technology, 2011):

- *Computer in the classroom:* Teachers are able to demonstrate a new lesson, present new material, illustrate how to use new programs, and show new websites.
- *Class website:* An easy way to display students' work is to create a web page designed for the class. Once a web page is designed, teachers can post homework assignments, student work, famous quotes, trivia games, etc.
- *Class blogs and wikis:* Blogs allow students to maintain a running dialogue, such as a journal, thoughts, ideas, and assignments that also provide for student comment and reflection. Wikis are more group focused and allow multiple members of the group to edit a single document and create a truly collaborative and carefully edited finished product.
- *Wireless classroom microphones:* With the help of microphones, students are able to hear their teachers more clearly and teachers no longer lose their voices at the end of the day.
- *Mobile devices:* Mobile devices such as smartphones can be used to enhance the experience in the classroom by providing

the possibility for teachers to get feedback.

- *Interactive whiteboards:* An interactive whiteboard that provides touch control of computer applications enhances the experience in the classroom by showing anything that can be on a computer screen. This not only aids in visual learning, but it is interactive so the students can draw, write, or manipulate images on the interactive whiteboard.

- *Online media:* Streamed video websites can be utilized to enhance a classroom lesson (e.g. United Streaming, Teacher Tube, etc.)

- *Digital games:* The field of educational games and serious games has been growing significantly over the last few years. Digital games are being provided as tools for the classroom and have a lot of positive feedback including higher motivation for students.

As far as software is concerned, an embarrassment of riches is offered. For years there have been attempts to create a digital library of computing containing educational material available (e.g., Grissom & al. 1998, Fox & al. 2002) resulting in at least two repositories: Computing and Information Technology Interactive Digital Educational Library (CITIDEL Repository, 2011), and OpenSeminar (2011). CITIDEL contains links to diverse communities offering several topics of computing (e.g., algorithm visualization, operating systems, and computer networks). It also offers a possibility to search or browse its material. OpenSeminar is, according to the website, "a web-based open courseware platform that enables instructors to collaborate on material for similar courses by sharing links to content. The end result of an OpenSeminar in a given subject is an expert-maintained repository or database of links to online information that is openly available on the Internet." In addition to the general repositories, there are specialized portals, such as AlgoViz.org, the Algorithm Visualization Portal (2009), which offers a variety of things to do: search for visualizations, read field reports, look for research papers, etc.

It seems that only a minor part of all educational tools are available in the repositories mentioned above. There are probably hundreds of different tools or similar: a search for "educational tool" in the ACM Digital Library produced 11 889 results (in November 2011). If we consider an educational tool in a wider meaning including teaching methods etc., we end up discussing such things as peer instruction (e.g., Porter & al. 2011), collaborative learning (e.g., Hamer & al. 2011), combining multiple pedagogies (e.g., Pollock & Harvey, 2011), storytelling and puzzles in teaching (e.g., Krishna Rao, 2006) as well as games (e.g., Sung, 2009; Nickel & Barnes, 2010), robots (e.g., Soule & Heckendorn, 2011), and virtual reality (e.g., Adams & Hotrop, 2008).

It is challenging to discover applications suitable for educational use. Repositories are useful but they cover only a minority of the tools available. It takes a lot of time to find out what else is on offer and whether there is anything reasonable available for the demand at hand.

3.3 CS education research

It is obvious that research should be the basis for teaching at the university level. However, it seems that adopting a new curriculum based on research results is not an easy task. Ni (2009) has conducted a study to determine factors influencing CS teachers' decision on whether to adopt a new CS curriculum. He found several factors influencing adoption summarizing that teachers' excitement in a new approach drives adoption, while more organizational or social issues inhibit adoption.

What kind of topics have been studied within CS education research, then? For example, the most recent proceedings of the *International Computing Education Research Workshop*, ICER 2011, includes scientific papers concerning collaborative learning, informal learning, CS1 (introduction to programming), research design, general-education computing, tools to support learning, and the ways in which students choose to study computing and specialize within computing. This list gives a nice view to the variety of topics concerned in CS education research.

Typical questions discussed are, e.g., "How to design courses and evaluate learning outcomes?", "How to combine course management, teaching and student learning in CS?", or "Why learning in certain topics is so difficult?". The first question concerning course design and evaluation of learning outcomes is a multidisciplinary problem, to which solutions have been tried to find in learning taxonomies. For example, Fuller & al. (2007) have been developing a CS-specific learning taxonomy—the Matrix Taxonomy—based on Bloom's taxonomy (Bloom & al., 1956) and revised Bloom's taxonomy (Anderson & al., 2000).

Further, the second question concerns learning management systems (LMS), such as Blackboard and Moodle. For example, Rößling & al. (2008) present an overview of CS specific on-line learning resources and provide guidance on how one could extend an LMS to include computer-based software tools. They also discuss an LMS that is extended specifically for CS education: Computing Augmented Learning Management System, CALMS. Another example of LMS use is presented by Rößling & al. (2010) discussing how to adapt Moodle to better support CS education.

The third question—why learning in certain topics is so difficult—can be exemplified with novice programming, which has been studied for decades. One of the latest papers trying to explain the reasons for difficulties in learning programming is Robins (2010). He introduces the learning edge momentum (LEM) effect as an alternative explanation to the earlier proposed division of students into two distinct populations of programmers and non-programmers.

There exists dozens of periodicals and conferences concerning computing and education. Their scientific quality varies from refereed to unreviewed, and sometimes it is difficult (or even impossible) to find out which one is at issue. Therefore, selecting the most valuable sources is really challenging. One possibility is to subscribe alerts on interesting topics (e.g., from ScienceDirect, 2011) and see if the alerts pay off. One option to keep informed is to sign up for the Csed-research mailing list (2008) which is intended to facilitate communication between CS education researchers.

Top publications

The are several conferences and scientific journals that publish papers on CS education research. The best journal for a researcher, in my opinion, is *Computer Science Education* (2011), whose own description says that the journal "presents information on educational research, current and prospective practices and techniques, the teaching experience, experiments with results, and educational software . . . [and c]overs fields of computer science, computer science and engineering, and software." The Computer Science Education journal publishes four issues per year containing 3–4 papers in each issue (15 altogether in year 2010). The leading conference is the earlier mentioned ICER (ACM, 2011b), which publishes papers concerning both learning and instruction (in year 2011 there were 18 papers accepted out of 47 submissions). Sheard & al. (2009) made an analysis of research into the teaching and learning of programming with a comparison of papers of six computing education conferences. Based on the results received, the ICER conference was ranked as the leading research conference for computing education.

From a teacher's point of view, the most interesting conferences are annually organized *SIGCSE Technical Symposium* (SIGCSE, 2011b) in the United States, and *Innovation and Technology in Computer Science Education* (aka ITiCSE; see SIGCSE, 2011a) in Europe or nearby. The SIGCSE Technical Symposium, as the conference itself announces, addresses problems common among educators working to develop, implement and/or evaluate computing programs, curricula, and courses. The symposium provides a forum for sharing new ideas for syllabi, laboratories, and other elements of teaching and pedagogy, at all levels of instruction. In the year 2011 there were 107 papers accepted (out of 315 submitted). The ITiCSE conference traditionally accepts submissions on the use of technology in supporting computer science teaching and learning, the practice of teaching computer science and computer science education research. This year there were 66 papers accepted (out of 169 submitted).

The number of papers of the previous publications is yearly about 200 in total. Teachers following these sources and reading at least a proportion of those papers yearly will be very knowledgeable and can be content.

4 CONCLUSIONS

In this paper we have discussed CS education and issues related to it. We noted that CS as a discipline has advanced a lot since the late 1950's when the first courses were taught. Likewise, learning theories have evolved during the past decades having an impact on CS education. Nevertheless, these changes have barely been noticeable in teachers' professional attainments—improvements are quite seldom exploited in everyday education. To ameliorate this situation, we presented a summary description of the knowledge that CS teachers should master, and introduced some options that are available for organizing tuition. Finally, we discussed the role of CS education research and stressed that it is essential for CS teachers to follow up the relevant literature in order to keep up-to-date.

5 REFERENCES

ACM (2001) Computing Curricula 2001: Computer Science, http://www.acm.org/education/education/education/curric_vols /cc2001.pdf

ACM (2005) Computing Curricula 2005: The overview report, http://www.acm.org/education/education/curric_vols/ CC2005-March06Final.pdf

ACM (2008) Computer Science Curriculum 2008: An Interim Revision of CS 2001, http://www.acm.org/education/curricula/ComputerScience2008.pdf

ACM (2011a) Curricula Recommendations, http://www.acm.org/education/curricula-recommendations

ACM (2011b) ICER conference, http://wp.acm.org/icer-conference/about/ (21.11.2011)

J.C. Adams and J. Hotrop (2008) Building an economical VR system for CS education. *SIGCSE Bull.* **40**(3), 148-152. http://doi.acm.org/10.1145/1597849.1384312

S. Alaoutinen and K. Smolander (2010) Are computer science students different learners?. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research (Koli Calling '10).* ACM, New York, NY, USA, 100-105. http://doi.acm.org/10.1145/1930464.1930482

AlgoViz.org, The Algorithm Visualization Portal (2009) http://algoviz.org/ (21.11.2011)

L.W. Anderson, D.R. Krathwohl, P.W. Airasian, K.A. Cruikshank, R.E. Mayer, P.R. Pintrich, J. Raths, and M.C. Wittrock (eds) (2000) *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives,* Abridged Edition. Allyn & Bacon.

B.S. Bloom, M.D. Engelhart, E.J. Furst, W.H. Hill, and D.R. Krathwohl (1956). *Taxonomy of educational objectives: the classification of educational goals; Handbook I: Cognitive Domain.* New York, Longmans, Green, 1956.

R. Boyle (2003) Who shall we put on the postage stamps?, *Technical Report 2003.10,* School of Computing, University of Leeds, http://www.engineering.leeds.ac.uk/computing/research/publications/reports/2003/2003_10.pdf

R. Boyle and M. Clark (2004) CS++: content is not enough. *SIGCSE Bull.* **36**(1), 422-426. http://doi.acm.org/10.1145/1028174.971443

J.S. Bruner (1967). *On knowing: Essays for the left hand.* Cambridge, Mass: Harvard University Press.

S. Card, T. Moran and A. Newell (1983) *The Psychology of Human-Computer Interaction,* Lawrence Erlbaum Associates, Hillsdale, NJ.

CITIDEL Repository (2011) http://www.citidel.org/ (21.11.2011)

Computer Science Education (2011)
http://www.tandf.co.uk/journals/NCSE

S.D. Conte, J.W. Hamblen, W.B. Kehl, S.O. Navarro, W.C. Rheinboldt, D.M. Young, Jr., and W.F. Atchinson (1965) An undergraduate program in computer science—preliminary recommendations. *Commun. ACM*, **8**(9),543–552.
http://doi.acm.org/10.1145/365559.366069

Csed-research mailing list (2008)
https://mailman2.u.washington.edu/mailman/listinfo/csed-research (21.11.2011)

D. Cukierman and D. McGee Thompson (2009) The academic enhancement program: encouraging students to learn about learning as part of their computing science courses. In *Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education (ITiCSE '09).* ACM, New York, NY, USA, 171-175. http://doi.acm.org/10.1145

P.J. Denning, D.E. Comer, D. Gries, M.C. Mulder, A. Tucker, A.J. Turner, and P.R. Young (1989) Computing as a discipline. *Commun. ACM*, **32**(1),9-23,
http://dx.doi.org.ezproxy.uef.fi:2048/10.1145/63238.63239

Educational technology (2011)
http://en.wikipedia.org/wiki/Educational_technology (21.11.2011)

E.A. Fox, H. Suleman and M. Luo (2002) "Building Digital Libraries Made Easy: Toward Open Digital Libraries", In *Proceedings of 5th International Conference on Asian Digital Libraries,* December 11-14, Singapore, 14-24.

U. Fuller, C.G. Johnson, T. Ahoniemi, D. Cukierman, I. Hernn-Losada, J. Jackova, E. Lahtinen, T.L. Lewis, D. McGee Thompson, C. Riedesel, and E. Thompson (2007) Developing a computer science-specific learning taxonomy. *SIGCSE Bull.* **39**(4), 152-170.

J. Gal-Ezer and D. Harel (1998) What (else) should CS educators know? *Commun. ACM*, **41**(9), 77-84.
http://doi.acm.org/10.1145/285070.285085

S. Grissom, D. Knox, E. Copperman, W. Dann, M. Goldweber, J. Hartman, M. Kuittinen, D. Mutchler, and N. Parlante (1998) "Developing a digital library of computer science teaching resources", *ACM SIGCUE Outlook,* **26**(4), 1-13.
http://doi.acm.org/10.1145/316572.358289

J. Hamer, A. Luxton-Reilly, H.C. Purchase, and J. Sheard (2011) Tools for "contributing student learning". *ACM Inroads* **2**(2), 78-91. http://doi.acm.org/10.1145/1963533.1963553

F. Haug (2011) Relevant Algorithm Animations/Visualizations (in Java). http://www.ansatt.hig.no/frodeh/algmet/animate.html

P.N. Johnson-Laird (1983) *Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness.* Cambridge: Cambridge University Press.

D.A. Kolb (1984) *Experiential Learning: Experience as the Source of Learning and Development.* Prentice-Hall, Inc., Englewood Cliffs, N.J.

M.R.K. Krishna Rao (2006) Storytelling and puzzles in a software engineering course. *SIGCSE Bull.* **38**(1), 418-422.
http://doi.acm.org/10.1145/1124706.1121472

J. Lave (1988) *Cognition in Practice: Mind, mathematics, and culture in everyday life.* Cambridge, UK: Cambridge University Press.

J. Lave and E. Wenger (1998) *Communities of Practice: Learning, Meaning, and Identity.* Cambridge University Press.

Learning-Theories.com — Knowledge Base and Webliography,
http://www.learning-theories.com/ (21.11.2011)

A.H. Maslow (1943) A Theory of Human Motivation. *Psychological Review,* **50**(4), 370-396.
http://psychclassics.yorku.ca/Maslow/motivation.htm

L. Ni (2009) What makes CS teachers change? Factors influencing CS teachers' adoption of curriculum innovations. *SIGCSE Bull.* **41**(1), 544-548.
http://doi.acm.org/10.1145/1539024.1509051

A. Nickel and T. Barnes (2010) Games for CS education: computer-supported collaborative learning and multiplayer games. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games (FDG '10).* ACM, New York, NY, USA, 274-276.
http://doi.acm.org/10.1145/1822348.1822391

OpenSeminar: A Community-based Platform for Collaborative Open Courseware. http://openseminar.org/se/ (21.11.2011)

I.P. Pavlov (1927) *Conditioned Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex.* Translated and Edited by G.V. Anrep. London: Oxford University Press.

J. Piaget (1928) La causalit chez l'enfant. *British Journal of Psychology,* **18**(3), 276-301

L. Pollock and T. Harvey (2011) Combining multiple pedagogies to boost learning and enthusiasm. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education (ITiCSE '11).* ACM, New York, NY, USA, 258-262.
http://doi.acm.org/10.1145/1999747.1999820

L. Porter, C. Bailey Lee, B. Simon, and D. Zingaro (2011) Peer instruction: do students really learn from peer discussion in computing? In *Proceedings of the seventh international workshop on Computing education research (ICER '11).* ACM, New York, NY, USA, 45-52.
http://doi.acm.org/10.1145/2016911.2016923

J.S. Renwick and C.B. Foltz (2011) Learning styles of information technology students. In *Proceedings of the 2011 conference on Information technology education (SIGITE '11).* ACM, New York, NY, USA, 313-314.
http://doi.acm.org/10.1145/2047594.2047679

A. Robins (2010) Learning edge momentum: a new account of outcomes in CS1, *Computer Science Education,* **20**(1), 37-71.
http://dx.doi.org/10.1080/08993401003612167

G. Rößling, M. Joy, A. Moreno, A. Radenski, L. Malmi, A. Kerren, T. Naps, R.J. Ross, M. Clancy, A. Korhonen, R. Oechsle, and J..

Velzquez Iturbide (2008) Enhancing learning management systems to better support computer science education. *SIGCSE Bull.* **40**(4), 142-166.
http://doi.acm.org/10.1145/1473195.1473239

G. Rößling, M. McNally, P. Crescenzi, A. Radenski, P. Ihantola, and M.G. Snchez-Torrubia (2010) Adapting Moodle to better support CS education. In *Proceedings of the 2010 ITiCSE working group reports on Working group reports (ITiCSE-WGR '10),* Alison Clear and Lori Russell Dag (Eds.). ACM, New York, NY, USA, 15-27.
http://doi.acm.org/10.1145/1971681.1971684

ScienceDirect (2011)
http://www.sciencedirect.com/science (21.11.2011)

J. Sheard, S. Simon, M. Hamilton, and J. Lönnberg (2009) Analysis of research into the teaching and learning of programming. In *Proceedings of the fifth international workshop on Computing education research workshop (ICER '09).* ACM, New York, NY, USA, 93-104.
http://doi.acm.org/10.1145/1584322.1584334

SIGCSE (2011a) ITiCSE Conferences,
http://www.sigcse.org/events/iticse
(21.11.2011)

SIGCSE (2011b) The SIGCSE Technical Symposium,
http://www.sigcse.org/events/symposia (21.11.2011)

T. Soule and R.B. Heckendorn (2011) COTSBots: computationally powerful, low-cost robots for Computer Science curriculums. *J. Comput. Sci. Coll.* **27**(1), 180-187.

Stanford CS Education Library (2006) http://cslibrary.stanford.edu/
(21.11.2011)

K. Sung (2009) Computer games and traditional CS courses. *Commun. ACM,* **52**(12), 74-78.
http://doi.acm.org/10.1145/1610252.1610273

J. Sweller (1988) "Cognitive load during problem solving: Effects on learning". *Cognitive Science,* **12**(2), 257-285.

M. Tedre (2011) Computing as a Science: A Survey of Competing Viewpoints. *Minds & Machines,* **21**(3), 361-387.
http://www.springerlink.com/content/v66j682n57602453/

TopUniversities (2011) Country guides, course information and university rankings for undergraduate degrees.
http://www.topuniversities.com/search/universities/computer%20science (21.11.2011)

Moving Threads and Parallel Thick Control Flows

Ville Leppñen
Department of Information Technology
University of Turku
Turku, Finland
ville.leppanen@it.utu.fi

Abstract. The purpose of this short paper is to challenge the reader to consider unconventional approaches. Two ideas concerning parallel computations are discussed in particular: Moving threads (instead of data), and parallel thick control flow. The author has had a major role in the development of both ideas. The parallel thick control flows represents actually a generalization of the ordinary sequential single-threaded computing to parallel flows with thickness in terms implicit threads. The unconventional approach stems from not having threads with a control flow but making each (parallel) control flow to have multiple implicit threads (thickness). Consequently, e.g. the program counter and call stack are then properties of each flow – not properties of each thread.

1 INTRODUCTION

Considering the theme of this conference 'CS I Like', I have always found theory of computer science very fascinating. If I would need to pick up one paper that has been very influential for me, then that paper would be Leslie Valiant's paper [13] on bridging model for parallel computing and its related handbook paper [12]. Those papers have lots of different kinds of messages. One message (unconventional view) is that one should move the focus in efficient routing from as small as possible routing time to throughput and as small as possible amortized cost per routed packet (ignoring the routing time that had been so dominating in the literature before). The bridging model that Valiant proposed had these routing related properties modelled – and several kind of architectures have been proposed ever since his paper to provide efficient throughput routing with as little parallel slackness used as possible.

Next, this paper will explain the concepts moving threads and parallel thick control flows. For more on parallel thick control flows, see [8]. The reader is challenged to consider the unconventional approach related to those concepts and the pros and cons of the approaches. In concluding section some remarks are given.

2 MOVING THREADS

We have developed a completely new kind of approach for mapping computations on multicore architectures [2, 3, 7, 10, 11] (some preliminary ideas appear in [5, 6]). We have also developed a simulator and a programming language for the approach and studied programming on it [9].

2.1 Idea

Instead of moving data read and write requests, we move extremely lightweight threads between the processor cores. Each processor core is coupled with memory module and parts of each memory module together form a virtual shared memory abstraction. Applications are written using a high-level language based on shared memory. As a consequence of moving threads instead of data we avoid all kinds of cache coherence problems. Other advantages are flexible and efficient creation of new threads. In our architecture, the challenge of having efficient implementation of an application reduces to mapping the used data so that the need to move threads is balanced with respect to the bandwidth of the communication lines. Writing an application to use lots of threads is rather easy (due to rich literature of parallel algorithms using shared memory abstraction). This method also eliminates the need for separate reply network and introduces a natural way to exploit locality without sacrificing the synchronicity of the PRAM model.

2.2 Overall architecture

An overview of our architectural framework is shown in Fig. 1. The system consists of c RISC-based *cores*, an inter-connection *network between the cores*, and a *main memory* system. Each core maintains a set of threads, can execute instructions from those, send and receive threads via the network, and has a cache memory for accessing a part of the main memory. Each core C_i "sees" a *unique fraction* of the main memory via its *data cache* – such memory locations are called *local* to C_i. Thus, if a thread residing at core C_i issues a memory instruction concerning some memory location local to core C_j, then the thread must be moved to C_j before executing the instruction. Moving a thread basically means moving the contents of its registers and program counter (as well as possibly next decoded instruction). The program, being executed by a thread to be moved, is not moved, since each core has an *instruction cache*, which contains fractions of all program codes being executed by the threads residing at that core.

Each memory location is local to only one core. Thus, there are no consistency problems, since there is no real replication of the contents of memory locations. Each memory location can be cached. The data caches of cores act as root access points into the main memory. In the framework, we do not specify how the main memory is organized – e.g. it can be partitioned into blocks. We neither do not fix the organization of the memory system – there can be multiple levels of caches. The mapping of memory locations into cores is not fixed in our architectural framework. We expect such a mapping to be balanced, but leave it open whether the mapping is static or dynamically set by the executed programs.

2.3 Single core and execution of moving threads programs

Each core maintains a dynamically varying set of threads by storing their register values in a *register file* and maintaining other information regarding them in a *thread pool*. A core extracts instructions from the threads (by using their program counter value) in

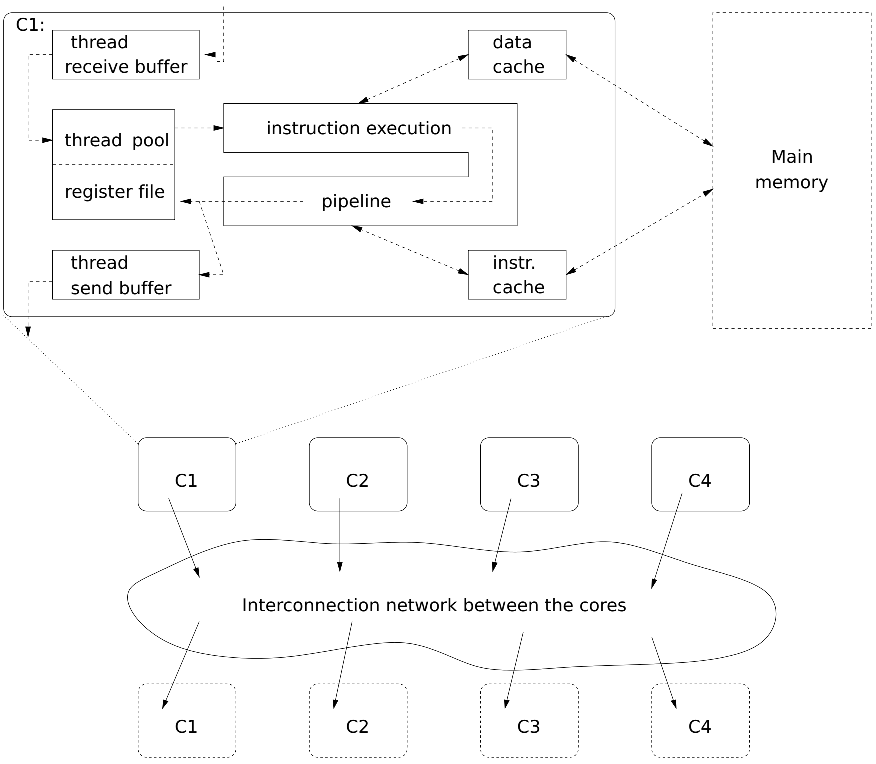


Figure 1: Overview of our multicore system.

its thread pool and injects such instruction into its instruction execution *pipeline*. None of instruction in the pipeline is a non-local memory instruction. The nature of next instruction is determined at the end of execution pipeline – thus, the need to move a thread is determined as early as possible. See Figure 2.

The goal is that each of the cores has $\Theta(X)$ threads to execute, and the threads are independent of each other – i.e. the core can take any of them and advance its execution. By taking an instruction cyclically from each thread, the core can wait for memory access taking a long time (and even tolerate the delays caused by moving the threads). The key to hide the memory (as well as network and other) delays is that the average number of threads X per core must be higher than the expected delay of executing a single

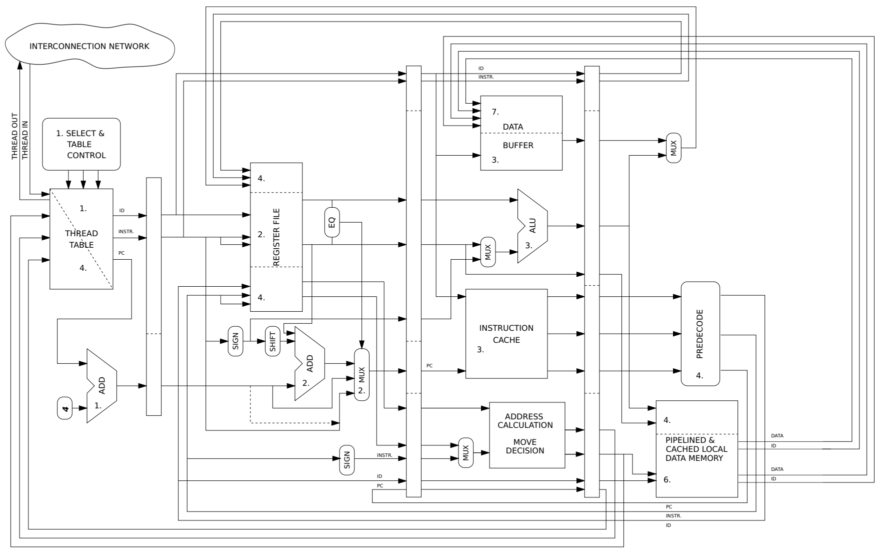


Figure 2: The datapath model of the moving threads processor core.

instruction from any thread.

The network connecting the cores is for moving threads between the cores. Thus, each core has separate *thread buffers* for sending and receiving threads. The received threads are moved into the thread pool of the receiving core, and respectively sending means removing a thread from the pool of the sending core.

The execution of all threads in the whole system is *synchronous*. The most strict interpretation of PRAM execution is that all threads execute synchronously stepwise – meaning that there is implicit synchronization after each step (i.e. atomic instruction). A less strict interpretation is that there is a separate synchronization instruction in the instruction set, and encountering such an instruction in the execution is treated as a barrier synchronization point (all threads pass over a barrier when all the threads have reached it). The instructions of a thread between two barrier synchronization points can be called as a superstep (notice that the length of superstep does not need to be static). The approach to the nature of execution synchrony is very crucial considering the semantics of programs and ease of programming. It is obvious that the more strict synchrony the easier to program but the more costly to implement. In our architectural framework, we do not specify how often the threads are synchronized, but we fix the architectural method of keeping the threads in synchrony. Our method is the *synchronization wave* method which can be seen to have been outlined already in the Fluent machine. The idea of synchronization wave is that a wave front separates two consecutive (super)steps. The wave front moves over an element (whether an interconnection node or an element related to the execution pipeline of a core) once it has arrived into the element via all input "links". Moving over a node means that the wave front is forwarded to all possible output "links" of the node.

3 THICK CONTROL FLOW

A software engineering perspective aiming at (reasoning about) correctness of parallel programs provides a fact supporting grouping of threads more tightly to each other and making each group to proceed (quite) synchronously. Reasoning about correctness of sequential programs or testing their execution has been successful, as one has been able see the execution of a program as a sequence of state transitions. In this respect, synchrony is a language-level or library-level mechanism to define states. The more the threads can do operations between states, the harder it will be to reason about the correctness of state transitions. If a huge number of threads can proceed at arbitrary speed, the program can be in very many different states. Such multithread programming is very difficult in correctness sense. Having only a few thick parallel synchronous control flows in the program makes it much easier to reason about the correctness, since such a mode of execution significantly reduces the amount of possible program states.

3.1 Idea

When a *thick control flow* (in terms of the number of threads) is executing a statement or an expression of a program, all the threads are considered to execute the same program element synchronously in parallel. Considering method calls, when a control flow with thickness t calls a method, the method is **not** called separately by each of the t threads, but the control flow calls it only once with t threads. A call stack is not related to each thread but to each of the parallel control flows, since threads do not have program counters – only control flows have program counters. Executing a branching statement means temporarily splitting a thick control flow into several other flows, which join after the branching statement.

The concept of thread is only implicit, although as language constructions statements for increasing/decreasing thickness are needed. The type system is extended with thickness. A thick variable is an array-like value having a thread-wise actual value. Method signatures naturally advance types with thickness, but non-thick types are also useful.

The concept of thick control flow makes the programmer to focus on co-operation of few parallel thick control flows instead of a huge number of parallel threads. The concept computation's state is promoted as a flow is seen to have a state (instead of each thread). The concept of state has been a central role in achieving correctness in sequential programs. The concept of thick control flows is related to data parallelism and stream computing. It is a natural generalization of ordinary imperative sequential programming.

The concept of thick control flows is a straightforward generalization of ordinary sequential program flows. A typical sequential program implicitly defines a lots of different kinds of control paths "through" the program. When a sequential program is run with a single thread, the thread follows exactly one of the possible control paths. From a semantical view point, one can consider that even a single-threaded program runs through all of its possible control paths in parallel. At each program statement involving a conditional branching of control (if-statements, switch-statements, loop condition checks, ...), the control of a single-threaded control flow can be considered to advance to all possible branches with thickness either one or zero threads – naturally we must require that the incoming thickness matches with the sum of outgoing thicknesses. Executing a control flow of zero thickness naturally has no effect (and can be ignored in practice).

3.2 Some considerations on program execution

Considering method calls, when a control flow with thickness t calls a method, the method is not called separately with each thread, but the control flow calls it only once with t threads. A call stack is not related to each thread but to each of the parallel control flows. Executing a branching statement can mean temporarily splitting a thick control flow into several other flows.

Originally, a program is considered to have a flow of thickness 1, measured conceptually in number of parallel threads. A method can be considered to have a thickness related to the calling flow's thickness. For creating a thick flow, we consider having to options: Either have a statement to dynamically set the thickness of the flow (by increasing/decreasing it), or have a block statement defining that the statements of the block are executed with a given thickness. We choose to support the latter option, *thick block*, as it is more naturally related to "thick", replicated variable declarations as well as the common idea of program stack.

A thick control flow of thickness t consists of t implicit threads. We assume them to have a unique identity expressed as an integer. Basically, we consider the identities to be between $0 \ldots t-1$, but it might be useful to have language constructions supporting other kind of indexing of implicit threads.

By a *replicated variable*, we mean a specifically declared variable within a thick block that has *a unique instance for each thread of the flow*. The degree of replication is dynamic, depending on the actual thickness of the flow. In practice, a replicated variable can be implemented as an array allocated from the memory.

The thick control flow concept clearly needs a stack-like structure for handling nested calls and nested thick blocks. However, as a flow can split into separate flows, the stack concept expands to a uniquely rooted tree-structure called *cactus tree*.

Nesting thick and ordinary block statements is meaningful and can be supported. Consider a situation where a thick block B_{out} of thickness t_{out} contains an inner thick block B_{in} of thickness t_{in}. A nested block is not executed thread-wise but flow-wise, and therefore considering the flow thickness, a flow executing the inner thick block has thickness t_{in} (instead of $t_{out} \times t_{in}$). There are issues concerning replicated variables, but those are not explained in this paper.

Consider a programming language constructs for branching statements like, `if-then-else`, `switch` ,and `case`. A control flow is executing each statement or expression of a program instead of a thread. When executing a branching statement with incoming control flow thickness of t_{in} and there are k possible branches with thicknesses $t_{out}^1, \ldots, t_{out}^k$, we simply require that $t_{in} = t_{out}^1 + \ldots + t_{out}^k$. Statements also involve joining the split control flows as the branches join – our thick control flow approach assumes that there will always be an implicit join at the end of a branching statement.

Considering semantics, the concept of thick control flows can be very helpful. All threads of a control flow can be seen to synchronously march through the (common) program code. When a flow is split into separate flows, it is perhaps best to consider that nothing is assumed about the advancing speed of the split flows. However, joining of different parts involves an implicit synchronization of the joined flows.

3.3 Function calls

Consider function calls, when a control flow with thickness t calls a method, the method is not called separately with each thread, but the control flow calls it only once with t implicit threads.

A function declaration needs to declare if it is meaningful to call a function with a thick flow (with e.g. a keyword 'rep' in the function signature). Recall that within a function one can change the thickness by using a thick block definition. The implicit flow thickness parameter can be seen as a possibility for the caller to define how many implicit threads should be used inside the function. For thick functions, it is meaningful to tie the thickness of passed values and return value to the thickness of calling flow. Notice that it is useful to be able to define non-replicated type in thick function's signature.

If the return value of a function is of replicated type, a normal return of a thick function call can be seen to return an array of values. It is also possible to consider some of the implicit threads to end their execution to an exception – then the thickness of normally returning control flow can be seen to reduce when some fraction of the incoming control flow has been separated by exceptions. The normal exception handling mechanisms (of e.g. OO languages) can be seen to later join the separated parts of the control flow with the original control flow.

3.4 Object-oriented features

The concept of parallel thick control flows can be extented to OO language constructions, too. The calling object can always be seen as the 0'th parameter of a function. In fact, if a call is applied to a replicated expression having object values, then a single method call can have multiple objects as callers!

The locking mechanism of objects can be preserved, but a lock should acquired by a thick control flow instead of an implicit thread within a thick control flow.

We do not further elaborate object-oriented features in this paper.

4 CONCLUSIONS

Hopefully, the two unconventional views discussed were stimulating. Determining the true benefits and drawbacks of moving threads and parallel thick control flows requires a lot of work. In case of the moving threads, it seems that threads should either (a) be seen as very simple (few registers per thread) and the true benefit comes from memory consistency, or (b) be seen as having lots of registers in which case the mechanism is an extension of the memory system to computing. The concept of parallel thick control flows is so new that its pros and cons are left open.

Finally, although it is exciting to look matters from a completely different viewpoint, the reader must be warned that selling the idea of the new viewpoint to the scientific community is often very difficult.

References

[1] M. Forsell and V. Leppänen. Supporting Concurrent Memory Access and Multioperations in Moving Threads CMPs. In *Proceedings of PDPTA 2010*, pages 377–383, 2010.

[2] M. Forsell and V. Leppänen. "Moving Threads: A Non-Conventional Approach for Mapping Computation to MP-SOC." In *Proceedings of the 2007 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'07)*, pages 232-238, Jun 2007.

[3] M. Forsell and V. Leppänen. Moving Threads Processor Architecture MTPA. *Journal of Supercomputing*, 57, pages 5–19, May 2011.

[4] J. Keller, C. Kessler, and J. Träff. *Practical PRAM Programming*. Wiley, 2001.

[5] V. Leppänen: *Studies on the Realization of PRAM*, PhD thesis, University of Turku, Department of Computer Science, TUCS Dissertation 3, November, 1996.

[6] V. Leppänen. "Balanced PRAM Simulations via Moving Threads and Hashing." *Journal of Universal Computer Science*, 4:8, 675–689, 1998.

[7] V. Leppänen, J.-M. Mäkelä and M. Forsell. *A RISC-Based Moving Tiny Threads Architecture*. Proceedings of 2011 Conference on Parallel and Distributed Processing Techniques and Appliations, PDPTA'11, II, CSREA Press, pp. 485–491, Jul 2011.

[8] V. Leppänen, M. Forsell and J.-M. Mäkelä. *Thick Control Flows: Introduction and Prospects,* Proceedings of 2011 Conference on Parallel and Distributed Processing Techniques and Appliations, PDPTA'11, II, CSREA Press, pp. 541–546, Jul 2011.

[9] J.M. Mäkelä and V. Leppänen. Towards programming on the moving threads architecture. In *CompSysTech '10: Proceedings of the International Conference on Computer Systems and Technologies*, pages 137–142. ACM Press, 2010.

[10] Mäkelä, J.-M., Leppänen, V. and Forsell M. *RISC-Based Moving Threads Multicore Architecture*. In Proceedings, 12th International Conference on Computer Systems and Technologies (CompSysTech'11), ACM Press ICPS Vol. 578, pages 51–56, Jun 2011.

[11] J. Paakkulainen, J.M. Mäkelä, V. Leppänen, and M. Forsell. Outline of risc-based core for multiprocessor on chip architecture supporting moving threads. In *CompSysTech '09: Proceedings of the International Conference on Computer Systems and Technologies*, pages 1–6. ACM Press, 2009.

[12] L.G. Valiant. "General Purpose Parallel Architectures". In *Handbook of Theoretical Computer Science, Ed. Jan van Leeuwen, Elsevier and MIT Press*, volume 1. Elsevier Science, 1990.

[13] L.G. Valiant. "A bridging model for parallel computation". Communication of the ACM, 33:8, pp. 103-111, 1990.

How to Invent and Prove a Result

Martti Penttonen
School of Computing
University of Eastern Finland
P.O. Box 1627, 70211 Kuopio, Finland
`martti.penttonen@uef.fi`

Abstract. In this talk we present, by a case, how a scientific result is achieved and proved, and through which phases does the research go. Our case concerns algorithmic research, in particular a routing algorithm. In other branches of computer science the stories may differ, but probably some similarity can be found.

1 INTRODUCTION

Each scientific result has a story behind, how the question arose, and how the result was achieved.

In mathematics, there is a long tradition of polishing the result and proof so that the story is hidden, at least for most. In computer science, results are often less sophisticated, and the history may even be explicitly written in the result and the proof.

In this paper, we tell the story of a result, a routing algorithm and its analysis. What is the structure of such a story? In our case, when the problem was algorithmic, the story went more or less as follows:

1. **Motivation.** What is the question? Why is that question interesting? Is it possible to get an answer, an approximation or an exact answer? Is it possible for us and now?

2. **First try.** Is there an obvious way to find a solution? Does it solve the problem?

3. **Experiment.** Test the obvious solution. Does it work at all? How good is it?

4. **Literature search.** Has somebody investigated the same problem or a rather similar problem? If yes, experiment with it, too.

5. **Reflection.** If the obvious or literature solutions did not work, why so? What is the core of the problem? What kind of methods might work.

6. **Invent and experiment.** If the idea did not work, return to the previous step. If it seems to work, continue.

7. **Clarify and prove** that the result holds.

8. **Test and tune.** Check the correctness and performance by practical tests, simulations. Simplify, recognize weaknessess, tune for performance.

9. **Write a report.**

Not all research problems are similar, but often the above pattern applies, at least in algorithm research.

2 THE PROBLEM

Our motivation came from the theory of parallel computation. Parallel computation would be easier, if processors had a shared memory, and if possible, with a low latency. If processors have local memory, shared memory could be implemented by fast parallel processor to processor communication. Optics would offer high bandwidth of communication, but collisions should be avoided. Under so called OCPC (Optically Connected Parallel Computer) condition packet sending succeeds only when there are no other packets trying to enter to the same target. Is there an algorithm that works efficiently under the OCPC assumption, preferrably simple and fast?

We found an algorithm thas seems suitable for the purpose. Afterwards, it seems very simple, almost naive. However, it required quite a lot of work, and quite a hard analysis. It offers a nice example, how algorithmic research may be done.

In a communication system, we want that anybody can communicate with anybody, but not all time. Usually one communicates with only one at a time, if any. The situation is not like Fig. 1 (a) but rather like Figure 1 (b).

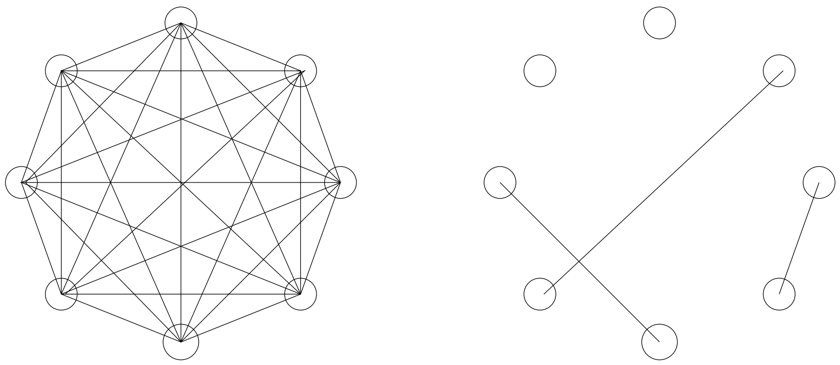


Figure 1: (a) All-to-all and (b) a more realistic routing task.

Further, it is not cost-efficient to have permanet, dedicated communication links between all pairs, but to use a shared communication resource. We assume that one agent, call it a processor, has a number of packets to send, but it can send only one packet at a time, see Figure 2 (a). On the other hand, it is natural to assume that processors cannot process many incoming messages at the same time. Hence, in Figure 2 (b) the sendings from 1 and 3 to 2 fail. This condition is called the *1-collision* condition. Colliding packets must be resent.

If packets are more or less randomly addressed, we can abstract our routing problem as an *h-relation*, in which each processor has at most h packets to route, and each processor is target of at most h packets. Our problem now is, how much time do we need to route all packets. If a processors has h packets, at least h time is needed, but do need more?

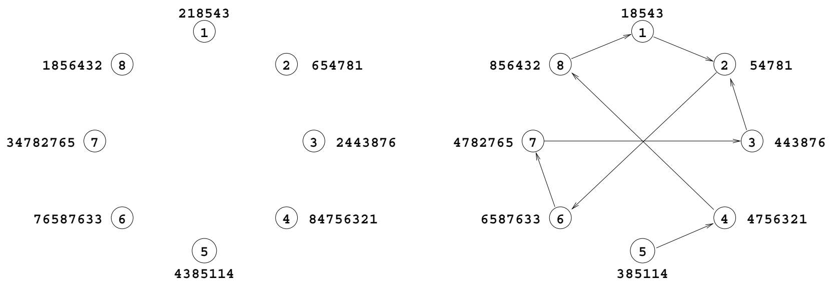


Figure 2: A step of a routing task.

3 A FIRST TRY AND IMPROVEMENTS

A good rule is that do it as simple as possible, but not more. The *greedy* principle means trying to move directly towards the solution, without worrying about the possible obstacles. In this case, the greedy algorithm is written as follows:

```
proc greedy
for all processors pardo
    while processor has packets do
        choose an unsent packet at random and try to send it
```

Figure 3: Greedy routing algorithm.

If packets are randomly addressed, which is natural unless we do not know anything special about the packets, the probability of successful sending, when there are p processors, is $(1-\frac{1}{p})^{p-1} \approx \frac{1}{e}$. Hence, $1/e$ fraction of packets may be successfully routed at every moment, and we might expect that e is enough time to route a packet.

Unfortunately the greedy algorithm does not fulfil our expectations. One soon observes that it ends up with a *livelock*: the same collision starts to repeat. Consider a situation when two processors both have only one packet, addressed to the same processor. They will collide for ever.

The fault in the greedy algorithm is that it forces to repeat the colliding step. That rule should be loosened somehow. A solution was proposed in [1]. Their idea is to start like greedy, but when the number of remaining packets has decreased from h to $k < h$, the packet is tried only with probability h/k. However, when the number of remaining packets has been halved, they restart recursively. With some more detail, their algorithm can be written as in Figure 4. In the algorithm, the recursion threshold is ϵ and the parameter α affects on the speed and success probability.

```
proc GGT($h$,$\epsilon$,$\alpha$)
for $i = 0$ to $\log_{1/(1-\epsilon)} h$ do
    for all processors pardo
        for $e(\epsilon h + max\{\sqrt{4\epsilon\alpha h \ln p}, 4\alpha \ln p\})/(1-\epsilon)$ times do
            choose an unsent packet $x$ at random
            attempt to send $x$ with probability # $unsent\ packets\ /\ h$
    $h := (1-\epsilon)h$
```

Figure 4: Geréb–Graus and Tsantilas algorithm.

In [1] it was proved that GGT algorithm works work-optimally, i.e. it routes any h-relation with $h \in \Omega(\log p \log\log p)$ in $O(h)$ time. But how good is it? How does it behave in practice?

In order to better see, how does the GGT algorithm behave, we wrote a simulator. With suitable values of parameters, the numbers of successful, non-tried, and remaining packets develop as depicted in Figure 5.

Looking at Figure 5, some questions arise. The sawblade pattern looks suspicious. If the number of unsent packets seems to

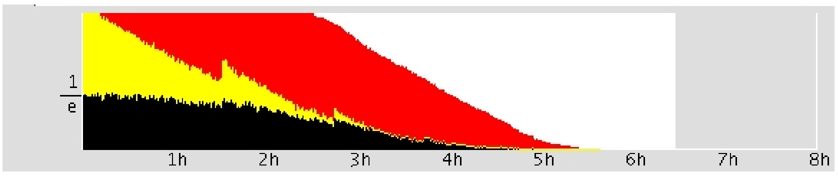


Figure 5: Routing with GGT.

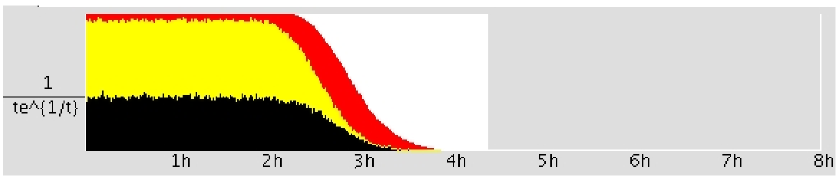


Figure 6: Routing with PCT.

decrease smoothly, why should the sending probability be changed abruptly? Shouldn't the sending probability remain the same or change smoothly. Figure 6 shows the throughput, when packets are sent with some probability less than 1. Call this protocol probabilistic constant thinning protocol, PCT. PCT seems to be faster than GGT.

Decreasing sending probability has two opposite effects. If you do not try to send, then the unsent packet does not arrive at its target. On the other hand, as fewer packets are being sent, the risk of collisions decreases and livelock is eliminated. If sending probability is $1/t$ ($t > 1$), expected routing time per packet would be $E(t) = te^{1/t}$. When $t > 1$ grows, the expectation $E(t)$ grows at first rather rather moderately: $E(1) = e \approx 2.72$, $E(2) \approx 3.3$, $E(3) \approx 4.2$ etc. At the beginning of routing, the greedy algorithm promises $E(1) = 2.72$ but it leads to livelock. If the sending probability is decreased to $1/1.1$, the expectation grows only to $E(1.1) = 2.73$ and we avoid the livelock.

The expected routing time calculated for random packets is not the whole truth. When the number of packet decreases, the distribution of packets becomes less and less random. This is seen in the long tail in the Figures 5 and 6. When only few packets remain, the probability of collisions grows higher. Therefore, when the number of remaining packet decreases to $\log p$, we let the the sending probability decrease. Even though the results in [4] are not directly applicable to our case, that paper was a good source of inspiration for understanding the tail phenomenon.

For practical reasons, instead of sending a packet with probability $1/t$, in CT algorithm of Figure 7, we send h packets in a time window of length *th*, which has roughly the same effect. In the algorithm, d is another constant, $1 < d < t$. In the routing progress curve of CT, in Figure 8 there are some discontinuities like in GGT, but smaller.

```
proc CT($h$,$h_0$,$t$,$d$)
for all processors pardo
    while $h > h_0$ do
        Try to transmit $h$ packets (if so many remain) in $\lceil th \rceil$ steps
        $h := (1 - e^{-1/d})h$
    while packets remain do
        Try to transmit the remaining packets in $\lceil th_0 \rceil$ steps
```

Figure 7: Constant thinning algorithm.

The curve in Figure 8 shows, that our "constant thinning" routing protocol performs quite well. Note that the best we can hope, is to route h packets in eh time. In [2] some generalized versions of the algorithm are presented.

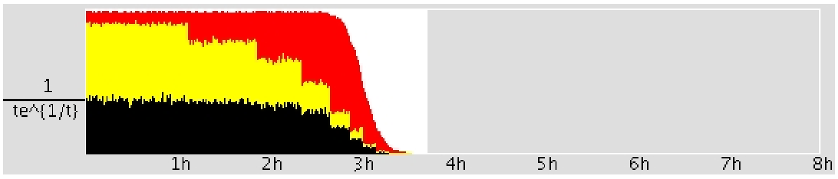


Figure 8: Routing with CT.

4 ANALYSIS

By experiments, literature and intuition, we came to a promising routing algorithm. But is it really as good as the experiments hint? Does it really route work-optimally, i.e. route h packets in $O(h)$ time? If so, under what assumptions?

The analysis of data communication protocols is often difficult. Still it is worth trying, not only for knowing the correctness for sure, but also for better understanding the algorithm and even improving it. We managed to prove the work optimality of a generalized version of the CT algorithm. It is found in [2] and we do not repeat the proof here. However, it is useful to recall some principles.

- *How does routing proceed in the large?* When a simulator is available, by experimenting with the parameters, one can observe that the main routing process and the tail routing process behave differently. In the main processing, thinning factor is important, in the tail processing, the choice of h_0 matters. Hence, the analysis should be divided in two parts.

- How to analyze a randomized algorithm? In simulations, we see that with good parameters, routing proceeds steadily. Can we trust on it? When we are dealing with expectations, often *Chernoff bounds* prove out to be useful. Intuitively, Chernoff bounds tell that it is not probable that a process deviates very much from the expectation. Some useful forms of Chernoff bounds [3] are:

  $$Pr(X < (1-\epsilon)E) \leq (\frac{e^{-\epsilon}}{(1-\epsilon)^{1-\epsilon}})^E$$

  $$Pr(X < (1-\epsilon)E) \leq e^{-\epsilon^2/2}$$

  $$Pr(X > (1+\epsilon)E) \leq (\frac{e^{\epsilon}}{(1+\epsilon)^{1+\epsilon}})^E$$

  $$Pr(X > (1+\epsilon)E) \leq e^{-\epsilon^2/3}$$

  and

  $$Pr(X > r) \leq 2^{-r} \ for \ r \geq 6E.$$

  By the first formula for $Pr(X < (1-\epsilon)E)$ we manage to prove that from round to round, the number of unsent packets decreased to a fraction, with high probability.

- *Progress from round to round?* By comparing packet numbers in two successive rounds, one can prove that the number of packets decreases in *geometric series*. Therefore, with high probability the number of packets reaches the $h_0 \in O(\log p)$ threshold in time that is proportional to $\log h$, where $h \in \Omega(\log p \log\log p)$ is the original number of packets.

In the tail, however, the decrease of packets is slower. One can see that when $h \leq h_0$ packets remain, expected sending time for a packet is eh_0/h. For all packets, this leads to a *harmonic series*, whose sum, also is in $O(\log p \log\log p)$. By the last form of Chernoff bound one can prove that $O(\log p \log\log p)$ time bound can be achieved with high probability.

- Throughout the analysis, some smaller mathematical tricks were needed.

In this way, we managed to prove that

for $h \in \Omega(\log p \log\log p)$, CT routes an h-relation work-optimally.

Furthermore, experiments show that by increasing the h "a little", the routing time comes close to $3h$. For a more general result, see [2]

5 CONCLUSIONS

By a well-known phrase, scientific results require some inspiration, and a lot of perspiration. We can affirm it. Perspiration is needed to familiarize with the problem and to develop suitable tools for experimentations. We used computer simulations for experimenting and learning the nature of the problem. Perspiration is also required for finding and learning right mathematical tools needed in the analysis. We found Chernoff bounds suitable for our purpose. Key word is *understanding the problem*. It inspires the intuition both in inventing a result, and in proving it.

References

[1] M. Geréb-Graus and T. Tsantilas. Efficient optical communication in parallel computers. In *SPAA'92, 4th Annual Symposium on Parallel Algorithms and Architectures, San Diego, California*, pages 41 – 48, June 1992.

[2] A. Kautonen, V. Leppänen and M. Penttonen. Thinning Protocols for Routing h-Relations Over Shared Media. *Journal of Parallel and Distributed Computing, 70(8):783-789, 2010*

[3] M. Mitzenmacher, E. Upfal: *Probability and Computing.* Cambridge University Press, 2005.

[4] M.S. Paterson and A. Srinivasan. Contention Resolution with Bounded Delay. In *Proceedings of 38th Annual IEEE Symposium on Foundations of Computer Science*, pages 104–113, 1995.

Martti Penttonen (ed.)

Computer Science I Like

Proceeding of Miniconference 4.11.2011

This proceedings is outcome of a miniconference held on 4.11.2011, at the University of Eastern Finland. The editor of this book, before retirement a month later, invited his former PhD students to give a talk under title Computer Science I Like. The point behind the title was that scientific research is not just work but also a vocation. As an event, the miniconference was a more or less regular scientific conference, although only one day long and with a wider scope. For students it offered a window into the world of research. Also, it was fun to meet. I thank all those, who made this miniconference and proceedings possible.

UNIVERSITY OF
EASTERN FINLAND

Publications of the University of Eastern Finland
Reports and Studies in Forestry and Natural Sciences

isbn 978-952-61-0635-9