PADERBORN UNIVERSITY

Seminar: Programmable Networks (WS24/25)
How to Read a Paper and Write a Report

Prof. Dr. Lin Wang
Computer Networks Group
Department of Computer Science
Paderborn University

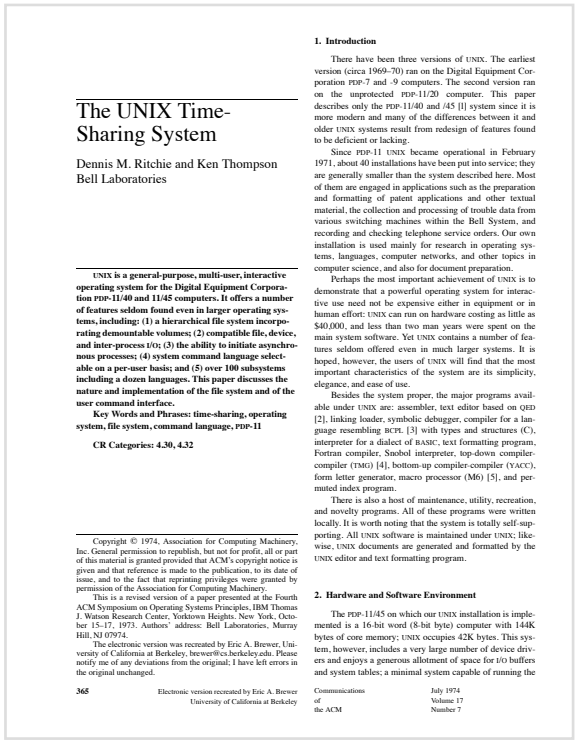
This lecture is on the key ingredients to succeed as a computer scientist.

Good coffee

Good sleep

Papers

A piece of writing to share original research work wirh other scientists and (maybe also) non-scientists



The UNIX Time-Sharing System

Dennis M. Ritchie and Ken Thompson
Bell Laboratories

UNIX is a general-purpose, multi-user, interactive operating system for the Digital Equipment Corporation PDP-11/40 and 11/45 computers. It offers a number of features seldom found even in larger operating systems, including: (1) a hierarchical file system incorporating demountable volumes; (2) compatible file, device, and inter-process I/O; (3) the ability to initiate asynchronous processes; (4) system command language selectable on a per-user basis; and (5) over 100 subsystems including a dozen languages. This paper discusses the nature and implementation of the file system and of the user command interface.

Key Words and Phrases: time-sharing, operating system, file system, command language, PDP-11

CR Categories: 4.30, 4.32

This is a revised version of a paper presented at the Fourth ACM Symposium on Operating Systems Principles, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, October 15–17, 1973. Authors' address: Bell Laboratories, Murray Hill, NJ 07974.

The electronic version was recreated by Eric A. Brewer, University of California at Berkeley, brewer@cs.berkeley.edu. Please notify me of any deviations from the original; I have left errors in the original unchanged.

1. Introduction

There have been three versions of UNIX. The earliest version (circa 1969–70) ran on the Digital Equipment Corporation PDP-7 and -9 computers. The second version ran on the unprotected PDP-11/20 computer. This paper describes only the PDP-11/40 and /45 [1] system since it is more modern and many of the differences between it and older UNIX systems result from redesign of features found to be deficient or lacking.

Since PDP-11 UNIX became operational in February 1971, about 40 installations have been put into service; they are generally smaller than the system described here. Most of them are engaged in applications such as the preparation and formatting of patent applications and other textual material, the collection and processing of trouble data from various switching machines within the Bell System, and recording and checking telephone service orders. Our own installation is used mainly for research in operating systems, languages, computer networks, and other topics in computer science, and also for document preparation.

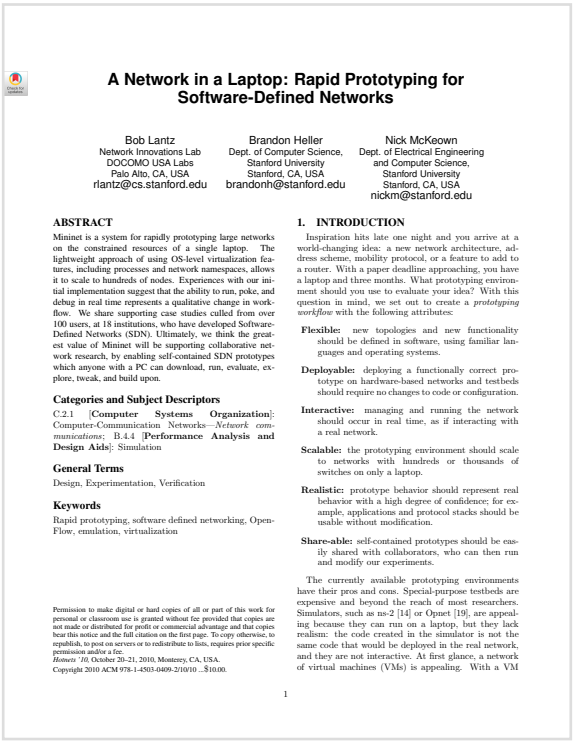
Perhaps the most important achievement of UNIX is to demonstrate that a powerful operating system for interactive use need not be expensive either in equipment or in human effort: UNIX can run on hardware costing as little as $40,000, and less than two man years were spent on the main system software. Yet UNIX contains a number of features seldom offered even in much larger systems. It is hoped, however, the users of UNIX will find that the most important characteristics of the system are its simplicity, elegance, and ease of use.

Besides the system proper, the major programs available under UNIX are: assembler, text editor based on QED [2], linking loader, symbolic debugger, compiler for a language resembling BCPL [3] with types and structures (C), interpreter for a dialect of BASIC, text formatting program, Fortran compiler, Snobol interpreter, top-down compiler-compiler (TMG) [4], bottom-up compiler-compiler (YACC), form letter generator, macro processor (M6) [5], and permuted index program.

There is also a host of maintenance, utility, recreation, and novelty programs. All of these programs were written locally. It is worth noting that the system is totally self-supporting. All UNIX software is maintained under UNIX; likewise, UNIX documents are generated and formatted by the UNIX editor and text formatting program.

2. Hardware and Software Environment

The PDP-11/45 on which our UNIX installation is implemented is a 16-bit word (8-bit byte) computer with 144K bytes of core memory; UNIX occupies 42K bytes. This system, however, includes a very large number of device drivers and enjoys a generous allotment of space for I/O buffers and system tables; a minimal system capable of running the



A Network in a Laptop: Rapid Prototyping for Software-Defined Networks

Bob Lantz
Network Innovations Lab
DOCOMO USA Labs
Palo Alto, CA, USA
rlantz@cs.stanford.edu

Brandon Heller
Dept. of Computer Science,
Stanford University
Stanford, CA, USA
brandonh@stanford.edu

Nick McKeown
Dept. of Electrical Engineering
and Computer Science,
Stanford University
Stanford, CA, USA
nickm@stanford.edu

ABSTRACT

Mininet is a system for rapidly prototyping large networks on the constrained resources of a single laptop. The lightweight approach of using OS-level virtualization features, including processes and network namespaces, allows it to scale to hundreds of nodes. Experiences with our initial implementation suggest that the ability to run, poke, and debug in real time represents a qualitative change in workflow. We share supporting case studies culled from over 100 users, at 18 institutions, who have developed Software-Defined Networks (SDN). Ultimately, we think the greatest value of Mininet will be supporting collaborative network research, by enabling self-contained SDN prototypes which anyone with a PC can download, run, evaluate, explore, tweak, and build upon.

Categories and Subject Descriptors

C.2.1 [**Computer Systems Organization**]: Computer-Communication Networks—*Network communications*; B.4.4 [**Performance Analysis and Design Aids**]: Simulation

General Terms

Design, Experimentation, Verification

Keywords

Rapid prototyping, software defined networking, OpenFlow, emulation, virtualization

Hotnets '10, October 20–21, 2010, Monterey, CA, USA.

1. INTRODUCTION

Inspiration hits late one night and you arrive at a world-changing idea: a new network architecture, address scheme, mobility protocol, or a feature to add to a router. With a paper deadline approaching, you have a laptop and three months. What prototyping environment should you use to evaluate your idea? With this question in mind, we set out to create a *prototyping workflow* with the following attributes:

Flexible: new topologies and new functionality should be defined in software, using familiar languages and operating systems.

Deployable: deploying a functionally correct prototype on hardware-based networks and testbeds should require no changes to code or configuration.

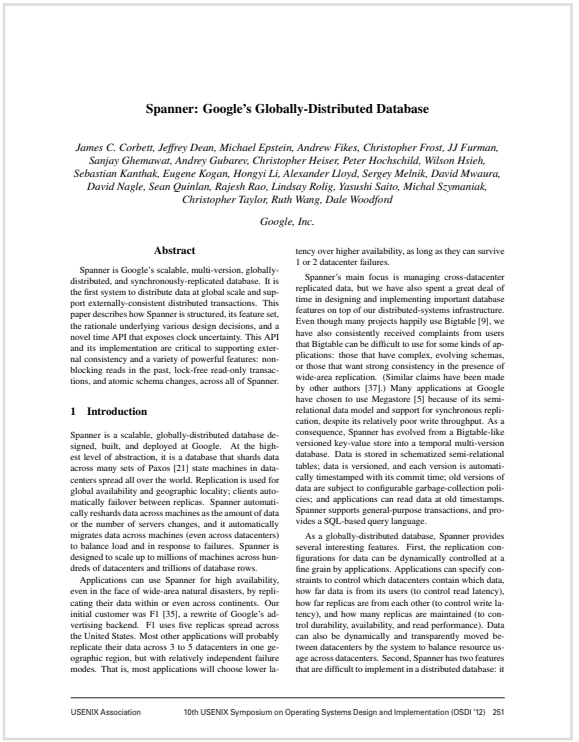
Interactive: managing and running the network should occur in real time, as if interacting with a real network.

Scalable: the prototyping environment should scale to networks with hundreds or thousands of switches on only a laptop.

Realistic: prototype behavior should represent real behavior with a high degree of confidence; for example, applications and protocol stacks should be usable without modification.

Share-able: self-contained prototypes should be easily shared with collaborators, who can then run and modify our experiments.

The currently available prototyping environments have their pros and cons. Special-purpose testbeds are expensive and beyond the reach of most researchers. Simulators, such as ns-2 [14] or Opnet [19], are appealing because they can run on a laptop, but they lack realism: the code created in the simulator is not the same code that would be deployed in the real network, and they are not interactive. At first glance, a network of virtual machines (VMs) is appealing. With a VM



Spanner: Google's Globally-Distributed Database

James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, Dale Woodford

Google, Inc.

Abstract

Spanner is Google's scalable, multi-version, globally-distributed, and synchronously-replicated database. It is the first system to distribute data at global scale and support externally-consistent distributed transactions. This paper describes how Spanner is structured, its feature set, the rationale underlying various design decisions, and a novel time API that exposes clock uncertainty. This API and its implementation are critical to supporting external consistency and a variety of powerful features: non-blocking reads in the past, lock-free read-only transactions, and atomic schema changes, across all of Spanner.

1 Introduction

Spanner is a scalable, globally-distributed database designed, built, and deployed at Google. At the highest level of abstraction, it is a database that shards data across many sets of Paxos [21] state machines in datacenters spread all over the world. Replication is used for global availability and geographic locality; clients automatically failover between replicas. Spanner automatically reshards data across machines as the amount of data or the number of servers changes, and it automatically migrates data across machines (even across datacenters) to balance load and in response to failures. Spanner is designed to scale up to millions of machines across hundreds of datacenters and trillions of database rows.

Applications can use Spanner for high availability, even in the face of wide-area natural disasters, by replicating their data within or even across continents. Our initial customer was F1 [35], a rewrite of Google's advertising backend. F1 uses five replicas spread across the United States. Most other applications will probably replicate their data across 3 to 5 datacenters in one geographic region, but with relatively independent failure modes. That is, most applications will choose lower latency over higher availability, as long as they can survive 1 or 2 datacenter failures.

Spanner's main focus is managing cross-datacenter replicated data, but we have also spent a great deal of time in designing and implementing important database features on top of our distributed-systems infrastructure. Even though many projects happily use Bigtable [9], we have also consistently received complaints from users that Bigtable can be difficult to use for some kinds of applications: those that have complex, evolving schemas, or those that want strong consistency in the presence of wide-area replication. (Similar claims have been made by other authors [37].) Many applications at Google have chosen to use Megastore [5] because of its semi-relational data model and support for synchronous replication, despite its relatively poor write throughput. As a consequence, Spanner has evolved from a Bigtable-like versioned key-value store into a temporal multi-version database. Data is stored in schematized semi-relational tables; data is versioned, and each version is automatically timestamped with its commit time; old versions of data are subject to configurable garbage-collection policies; and applications can read data at old timestamps. Spanner supports general-purpose transactions, and provides a SQL-based query language.

As a globally-distributed database, Spanner provides several interesting features. First, the replication configurations for data can be dynamically controlled at a fine grain by applications. Applications can specify constraints to control which datacenters contain which data, how far data is from its users (to control read latency), how far replicas are from each other (to control write latency), and how many replicas are maintained (to control durability, availability, and read performance). Data can also be dynamically and transparently moved between datacenters by the system to balance resource usage across datacenters. Second, Spanner has two features that are difficult to implement in a distributed database: it


Reading and
understanding is hard
[pexels.com]

Stages of reading a scientific paper

Optimism

Regret

Contradiction

Determination

Genuine contemplation of career choices

Fear

Concerned

Panic

Rage

Stages of reading a scientific paper

Optimism

Regret

Contradiction

Determination

Genuine contemplation of career choices

Fear

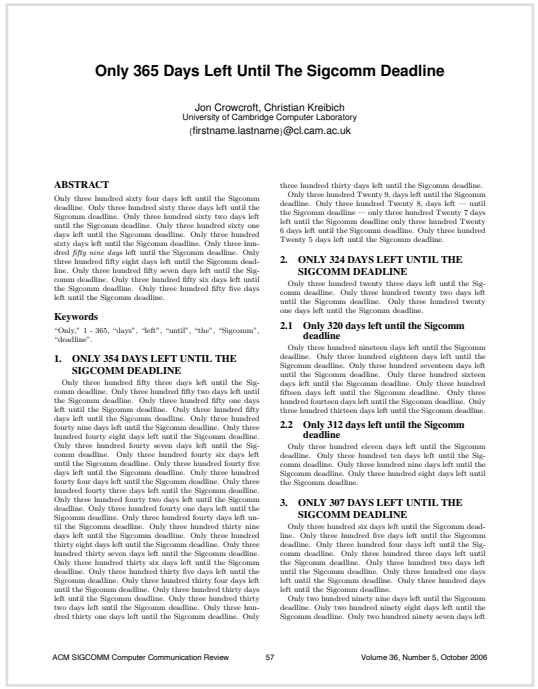
Concerned

Panic

Rage

Do not worry, we have all been through these stages

Papers can be fun



Only 365 Days Left Until The Sigcomm Deadline

Jon Crowcroft, Christian Kreibich
University of Cambridge Computer Laboratory
(firstname.lastname)@cl.cam.ac.uk

ABSTRACT

Only three hundred sixty four days left until the Sigcomm deadline. Only three hundred sixty three days left until the Sigcomm deadline. Only three hundred sixty two days left until the Sigcomm deadline. Only three hundred sixty one days left until the Sigcomm deadline. Only three hundred sixty days left until the Sigcomm deadline. Only three hundred *fifty nine days* left until the Sigcomm deadline. Only three hundred fifty eight days left until the Sigcomm deadline. Only three hundred fifty seven days left until the Sigcomm deadline. Only three hundred fifty six days left until the Sigcomm deadline. Only three hundred fifty five days left until the Sigcomm deadline.

Keywords

"Only," 1 - 365, "days", "left", "until", "the", "Sigcomm", "deadline".

1. ONLY 354 DAYS LEFT UNTIL THE SIGCOMM DEADLINE

Only three hundred fifty three days left until the Sigcomm deadline. Only three hundred fifty two days left until the Sigcomm deadline. Only three hundred fifty one days left until the Sigcomm deadline. Only three hundred fifty days left until the Sigcomm deadline. Only three hundred fourty nine days left until the Sigcomm deadline. Only three hundred fourty eight days left until the Sigcomm deadline. Only three hundred fourty seven days left until the Sigcomm deadline. Only three hundred fourty six days left until the Sigcomm deadline. Only three hundred fourty five days left until the Sigcomm deadline. Only three hundred fourty four days left until the Sigcomm deadline. Only three hundred fourty three days left until the Sigcomm deadline. Only three hundred fourty two days left until the Sigcomm deadline. Only three hundred fourty one days left until the Sigcomm deadline. Only three hundred fourty days left until the Sigcomm deadline. Only three hundred thirty nine days left until the Sigcomm deadline. Only three hundred thirty eight days left until the Sigcomm deadline. Only three hundred thirty seven days left until the Sigcomm deadline. Only three hundred thirty six days left until the Sigcomm deadline. Only three hundred thirty five days left until the Sigcomm deadline. Only three hundred thirty four days left until the Sigcomm deadline. Only three hundred thirty days left until the Sigcomm deadline. Only three hundred thirty two days left until the Sigcomm deadline. Only three hundred thirty one days left until the Sigcomm deadline. Only three hundred thirty days left until the Sigcomm deadline.

Only three hundred Twenty 9, days left until the Sigcomm deadline. Only three hundred Twenty 8, days left — until the Sigcomm deadline — only three hundred Twenty 7 days left until the Sigcomm deadline only three hundred Twenty 6 days left until the Sigcomm deadline. Only three hundred Twenty 5 days left until the Sigcomm deadline.

2. ONLY 324 DAYS LEFT UNTIL THE SIGCOMM DEADLINE

Only three hundred twenty three days left until the Sigcomm deadline. Only three hundred twenty two days left until the Sigcomm deadline. Only three hundred twenty one days left until the Sigcomm deadline.

2.1 Only 320 days left until the Sigcomm deadline

Only three hundred nineteen days left until the Sigcomm deadline. Only three hundred eighteen days left until the Sigcomm deadline. Only three hundred seventeen days left until the Sigcomm deadline. Only three hundred sixteen days left until the Sigcomm deadline. Only three hundred fifteen days left until the Sigcomm deadline. Only three hundred fourteen days left until the Sigcomm deadline. Only three hundred thirteen days left until the Sigcomm deadline.

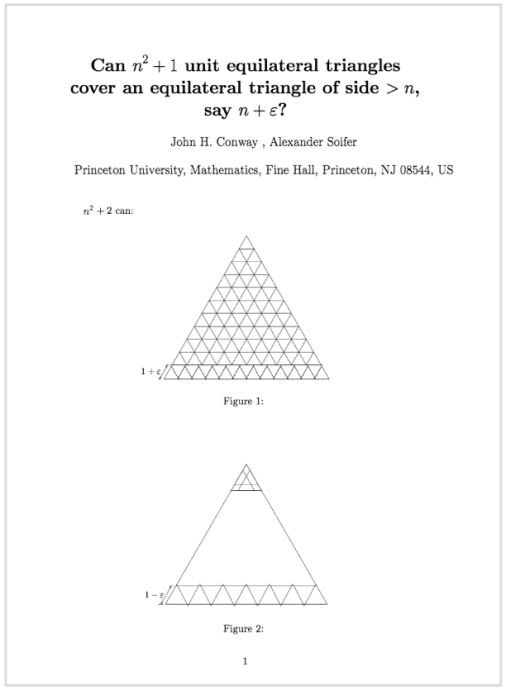
2.2 Only 312 days left until the Sigcomm deadline

Only three hundred eleven days left until the Sigcomm deadline. Only three hundred ten days left until the Sigcomm deadline. Only three hundred nine days left until the Sigcomm deadline. Only three hundred eight days left until the Sigcomm deadline.

3. ONLY 307 DAYS LEFT UNTIL THE SIGCOMM DEADLINE

Only three hundred six days left until the Sigcomm deadline. Only three hundred five days left until the Sigcomm deadline. Only three hundred four days left until the Sigcomm deadline. Only three hundred three days left until the Sigcomm deadline. Only three hundred two days left until the Sigcomm deadline. Only three hundred one days left until the Sigcomm deadline. Only three hundred days left until the Sigcomm deadline.

Only two hundred ninety nine days left until the Sigcomm deadline. Only two hundred ninety eight days left until the Sigcomm deadline. Only two hundred ninety seven days left



Can n^2+1 unit equilateral triangles cover an equilateral triangle of side $> n$, say $n+\varepsilon$?

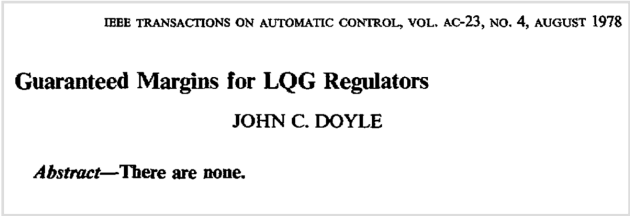
John H. Conway , Alexander Soifer

Princeton University, Mathematics, Fine Hall, Princeton, NJ 08544, US

n^2+2 can:

Figure 1:

Figure 2:

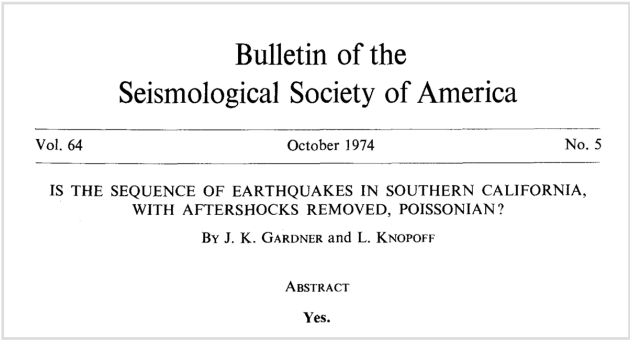


IEEE TRANSACTIONS ON AUTOMATIC CONTROL, VOL. AC-23, NO. 4, AUGUST 1978

Guaranteed Margins for LQG Regulators

JOHN C. DOYLE

***Abstract*—There are none.**



Bulletin of the
Seismological Society of America

Vol. 64 October 1974 No. 5

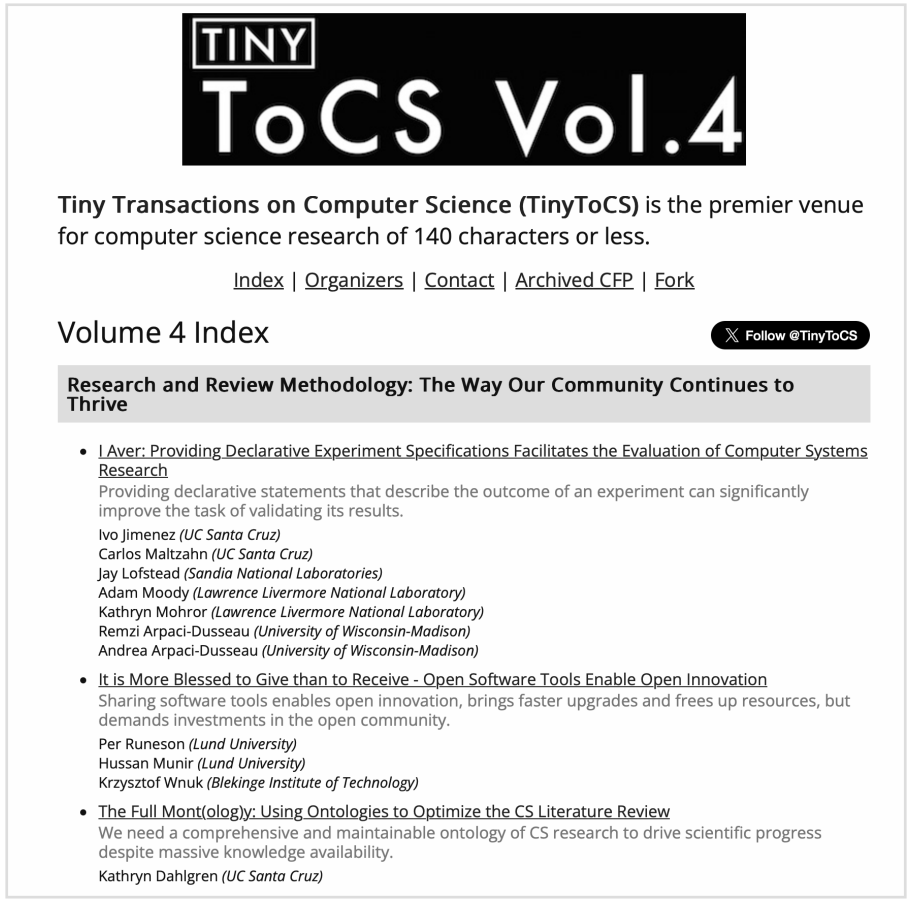
IS THE SEQUENCE OF EARTHQUAKES IN SOUTHERN CALIFORNIA, WITH AFTERSHOCKS REMOVED, POISSONIAN?

BY J. K. GARDNER and L. KNOPOFF

ABSTRACT

Yes.

For big fans of reading short papers


TINY ToCS Vol.4

Tiny Transactions on Computer Science (TinyToCS) is the premier venue for computer science research of 140 characters or less.

Index | Organizers | Contact | Archived CFP | Fork

Volume 4 Index

Follow @TinyToCS

Research and Review Methodology: The Way Our Community Continues to Thrive

- I Aver: Providing Declarative Experiment Specifications Facilitates the Evaluation of Computer Systems Research
  Providing declarative statements that describe the outcome of an experiment can significantly improve the task of validating its results.
  Ivo Jimenez *(UC Santa Cruz)*
  Carlos Maltzahn *(UC Santa Cruz)*
  Jay Lofstead *(Sandia National Laboratories)*
  Adam Moody *(Lawrence Livermore National Laboratory)*
  Kathryn Mohror *(Lawrence Livermore National Laboratory)*
  Remzi Arpaci-Dusseau *(University of Wisconsin-Madison)*
  Andrea Arpaci-Dusseau *(University of Wisconsin-Madison)*
- It is More Blessed to Give than to Receive - Open Software Tools Enable Open Innovation
  Sharing software tools enables open innovation, brings faster upgrades and frees up resources, but demands investments in the open community.
  Per Runeson *(Lund University)*
  Hussan Munir *(Lund University)*
  Krzysztof Wnuk *(Blekinge Institute of Technology)*
- The Full Mont(olog)y: Using Ontologies to Optimize the CS Literature Review
  We need a comprehensive and maintainable ontology of CS research to drive scientific progress despite massive knowledge availability.
  Kathryn Dahlgren *(UC Santa Cruz)*

http://tinytocs.org/

Papers are fun

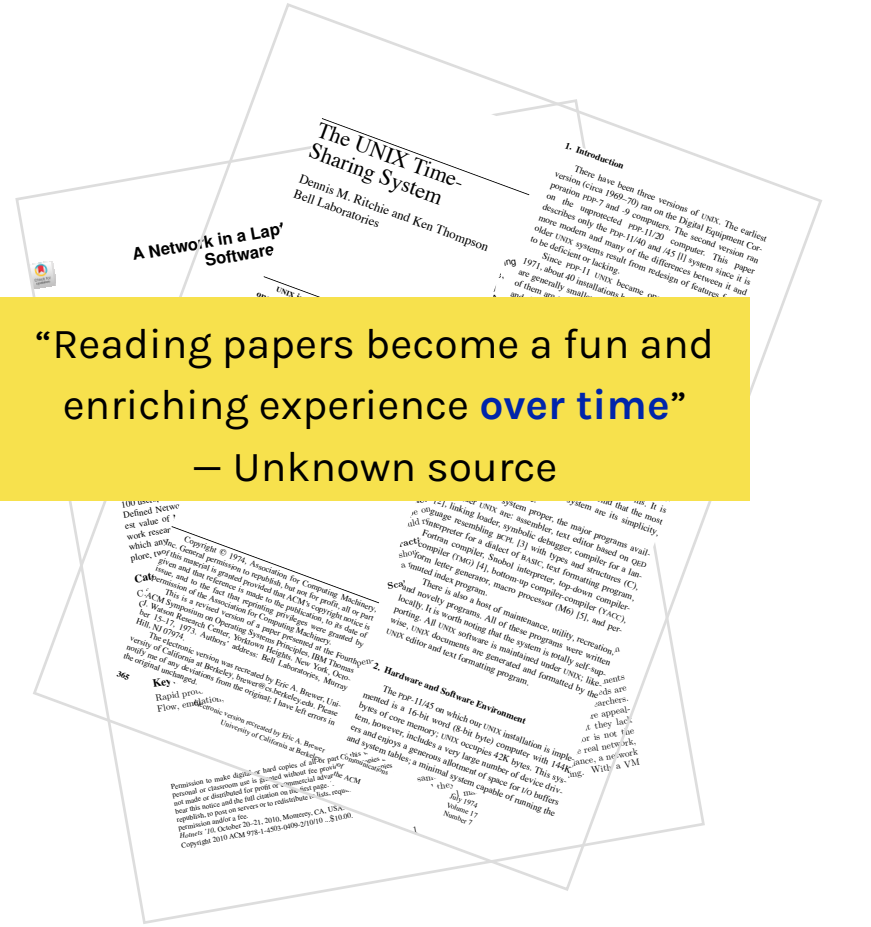
Written with a specific audience in mind

Context or background is key

- Richard Feynman answering a question on magnetism: https://www.youtube.com/watch?v=M00r930Sn_8

Strengthen your background

- Read voraciously
- No shortcuts!



"Reading papers become a fun and enriching experience **over time**"
— Unknown source

Prepare

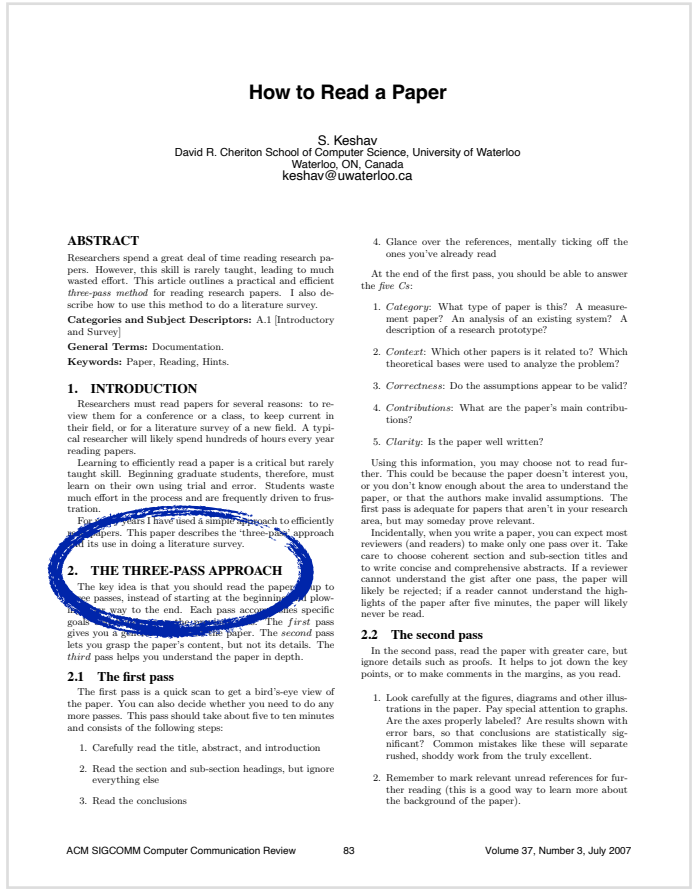
Environment

- Place: cafe, library, ...
- Ambience: music, quiet, ...
- Medium: printout, tablet, laptop, ...

Set expectations

- Presentation?
- Critique?
- Leisurely read?

Start by reading a paper



How to Read a Paper

S. Keshav
David R. Cheriton School of Computer Science, University of Waterloo
Waterloo, ON, Canada
keshav@uwaterloo.ca

ABSTRACT

Researchers spend a great deal of time reading research papers. However, this skill is rarely taught, leading to much wasted effort. This article outlines a practical and efficient *three-pass method* for reading research papers. I also describe how to use this method to do a literature survey.

Categories and Subject Descriptors: A.1 [Introductory and Survey]

General Terms: Documentation.

Keywords: Paper, Reading, Hints.

1. INTRODUCTION

Researchers must read papers for several reasons: to review them for a conference or a class, to keep current in their field, or for a literature survey of a new field. A typical researcher will likely spend hundreds of hours every year reading papers.

Learning to efficiently read a paper is a critical but rarely taught skill. Beginning graduate students, therefore, must learn on their own using trial and error. Students waste much effort in the process and are frequently driven to frustration.

For [illegible] years I have used a simple approach to efficiently read papers. This paper describes the 'three-pass' approach and its use in doing a literature survey.

2. THE THREE-PASS APPROACH

The key idea is that you should read the paper in up to three passes, instead of starting at the beginning and plowing your way to the end. Each pass accomplishes specific goals and builds upon the results of the previous pass: The *first* pass gives you a general idea about the paper. The *second* pass lets you grasp the paper's content, but not its details. The *third* pass helps you understand the paper in depth.

2.1 The first pass

The first pass is a quick scan to get a bird's-eye view of the paper. You can also decide whether you need to do any more passes. This pass should take about five to ten minutes and consists of the following steps:

1. Carefully read the title, abstract, and introduction
2. Read the section and sub-section headings, but ignore everything else
3. Read the conclusions
4. Glance over the references, mentally ticking off the ones you've already read

At the end of the first pass, you should be able to answer the *five Cs*:

1. *Category*: What type of paper is this? A measurement paper? An analysis of an existing system? A description of a research prototype?
2. *Context*: Which other papers is it related to? Which theoretical bases were used to analyze the problem?
3. *Correctness*: Do the assumptions appear to be valid?
4. *Contributions*: What are the paper's main contributions?
5. *Clarity*: Is the paper well written?

Using this information, you may choose not to read further. This could be because the paper doesn't interest you, or you don't know enough about the area to understand the paper, or that the authors make invalid assumptions. The first pass is adequate for papers that aren't in your research area, but may someday prove relevant.

Incidentally, when you write a paper, you can expect most reviewers (and readers) to make only one pass over it. Take care to choose coherent section and sub-section titles and to write concise and comprehensive abstracts. If a reviewer cannot understand the gist after one pass, the paper will likely be rejected; if a reader cannot understand the highlights of the paper after five minutes, the paper will likely never be read.

2.2 The second pass

In the second pass, read the paper with greater care, but ignore details such as proofs. It helps to jot down the key points, or to make comments in the margins, as you read.

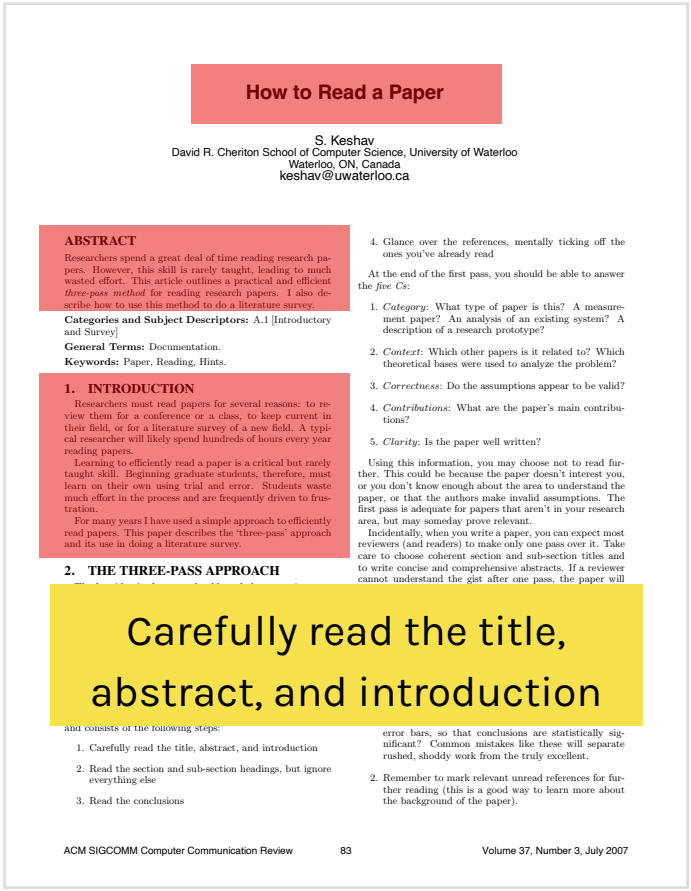
1. Look carefully at the figures, diagrams and other illustrations in the paper. Pay special attention to graphs. Are the axes properly labeled? Are results shown with error bars, so that conclusions are statistically significant? Common mistakes like these will separate rushed, shoddy work from the truly excellent.
2. Remember to mark relevant unread references for further reading (this is a good way to learn more about the background of the paper).

http://ccr.sigcomm.org/online/files/p83-keshavA.pdf

First pass

Obtain a bird's eye view of the paper

First pass



How to Read a Paper

S. Keshav
David R. Cheriton School of Computer Science, University of Waterloo
Waterloo, ON, Canada
keshav@uwaterloo.ca

ABSTRACT

Researchers spend a great deal of time reading research papers. However, this skill is rarely taught, leading to much wasted effort. This article outlines a practical and efficient *three-pass method* for reading research papers. I also describe how to use this method to do a literature survey.

Categories and Subject Descriptors: A.1 [Introductory and Survey]
General Terms: Documentation.
Keywords: Paper, Reading, Hints.

1. INTRODUCTION

Researchers must read papers for several reasons: to review them for a conference or a class, to keep current in their field, or for a literature survey of a new field. A typical researcher will likely spend hundreds of hours every year reading papers.

Learning to efficiently read a paper is a critical but rarely taught skill. Beginning graduate students, therefore, must learn on their own using trial and error. Students waste much effort in the process and are frequently driven to frustration.

For many years I have used a simple approach to efficiently read papers. This paper describes the 'three-pass' approach and its use in doing a literature survey.

2. THE THREE-PASS APPROACH

Carefully read the title, abstract, and introduction

and consists of the following steps:

1. Carefully read the title, abstract, and introduction
2. Read the section and sub-section headings, but ignore everything else
3. Read the conclusions
4. Glance over the references, mentally ticking off the ones you've already read

At the end of the first pass, you should be able to answer the *five Cs*:

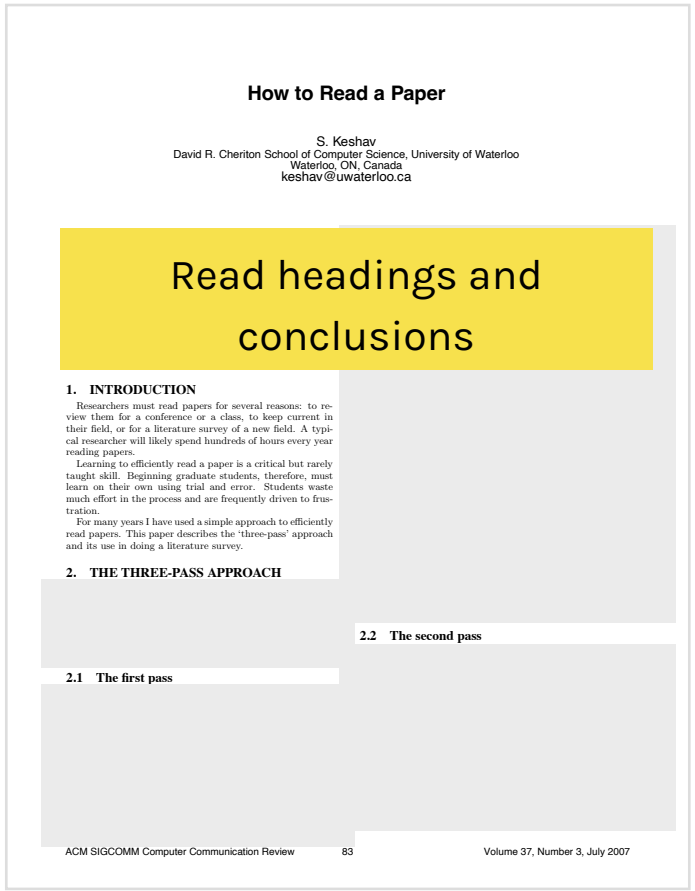
1. *Category*: What type of paper is this? A measurement paper? An analysis of an existing system? A description of a research prototype?
2. *Context*: Which other papers is it related to? Which theoretical bases were used to analyze the problem?
3. *Correctness*: Do the assumptions appear to be valid?
4. *Contributions*: What are the paper's main contributions?
5. *Clarity*: Is the paper well written?

Using this information, you may choose not to read further. This could be because the paper doesn't interest you, or you don't know enough about the area to understand the paper, or that the authors make invalid assumptions. The first pass is adequate for papers that aren't in your research area, but may someday prove relevant.

Incidentally, when you write a paper, you can expect most reviewers (and readers) to make only one pass over it. Take care to choose coherent section and sub-section titles and to write concise and comprehensive abstracts. If a reviewer cannot understand the gist after one pass, the paper will

error bars, so that conclusions are statistically significant? Common mistakes like these will separate rushed, shoddy work from the truly excellent.

2. Remember to mark relevant unread references for further reading (this is a good way to learn more about the background of the paper).



How to Read a Paper

S. Keshav
David R. Cheriton School of Computer Science, University of Waterloo
Waterloo, ON, Canada
keshav@uwaterloo.ca

Read headings and conclusions

1. INTRODUCTION

Researchers must read papers for several reasons: to review them for a conference or a class, to keep current in their field, or for a literature survey of a new field. A typical researcher will likely spend hundreds of hours every year reading papers.

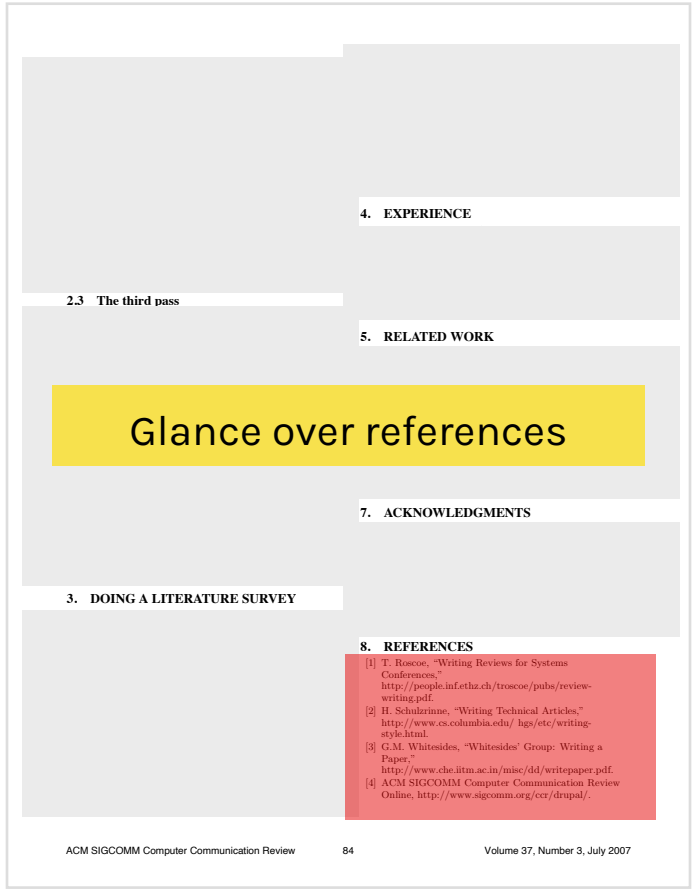
Learning to efficiently read a paper is a critical but rarely taught skill. Beginning graduate students, therefore, must learn on their own using trial and error. Students waste much effort in the process and are frequently driven to frustration.

For many years I have used a simple approach to efficiently read papers. This paper describes the 'three-pass' approach and its use in doing a literature survey.

2. THE THREE-PASS APPROACH

2.1 The first pass

2.2 The second pass



2.3 The third pass

3. DOING A LITERATURE SURVEY

4. EXPERIENCE

5. RELATED WORK

Glance over references

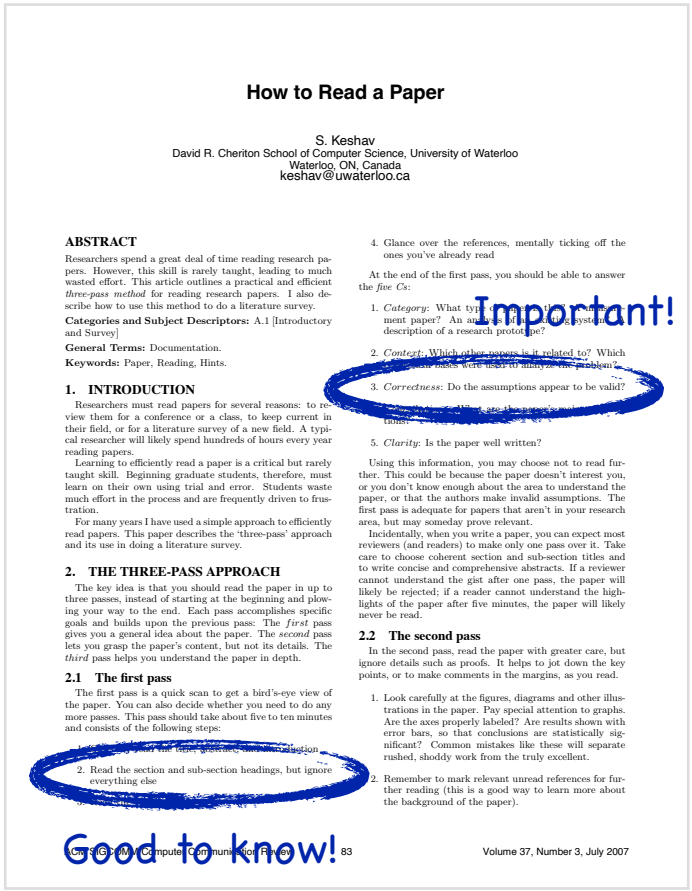
7. ACKNOWLEDGMENTS

8. REFERENCES

[1] T. Roscoe, "Writing Reviews for Systems Conferences," http://people.inf.ethz.ch/troscoe/pubs/review-writing.pdf.
[2] H. Schulzrinne, "Writing Technical Articles," http://www.cs.columbia.edu/ hgs/etc/writing-style.html.
[3] G.M. Whitesides, "Whitesides' Group: Writing a Paper," http://www.che.iitm.ac.in/misc/dd/writepaper.pdf.
[4] ACM SIGCOMM Computer Communication Review Online, http://www.sigcomm.org/ccr/drupal/.

Second pass

Read with greater care, but ignore details



How to Read a Paper

S. Keshav
David R. Cheriton School of Computer Science, University of Waterloo
Waterloo, ON, Canada
keshav@uwaterloo.ca

ABSTRACT

Researchers spend a great deal of time reading research papers. However, this skill is rarely taught, leading to much wasted effort. This article outlines a practical and efficient *three-pass method* for reading research papers. I also describe how to use this method to do a literature survey.

Categories and Subject Descriptors: A.1 [Introductory and Survey]

General Terms: Documentation.

Keywords: Paper, Reading, Hints.

1. INTRODUCTION

Researchers must read papers for several reasons: to review them for a conference or a class, to keep current in their field, or for a literature survey of a new field. A typical researcher will likely spend hundreds of hours every year reading papers.

Learning to efficiently read a paper is a critical but rarely taught skill. Beginning graduate students, therefore, must learn on their own using trial and error. Students waste much effort in the process and are frequently driven to frustration.

For many years I have used a simple approach to efficiently read papers. This paper describes the 'three-pass' approach and its use in doing a literature survey.

2. THE THREE-PASS APPROACH

The key idea is that you should read the paper in up to three passes, instead of starting at the beginning and plowing your way to the end. Each pass accomplishes specific goals and builds upon the previous pass: The *first* pass gives you a general idea about the paper. The *second* pass lets you grasp the paper's content, but not its details. The *third* pass helps you understand the paper in depth.

2.1 The first pass

The first pass is a quick scan to get a bird's-eye view of the paper. You can also decide whether you need to do any more passes. This pass should take about five to ten minutes and consists of the following steps:

2. Read the section and sub-section headings, but ignore everything else

4. Glance over the references, mentally ticking off the ones you've already read

At the end of the first pass, you should be able to answer the *five Cs*:

1. *Category:* What type of paper is this? A measurement paper? An analysis of a piece of software? A description of a research prototype?

2. *Context:* Which other papers is it related to? Which theoretical bases were used to analyze the problem?

3. *Correctness:* Do the assumptions appear to be valid?

5. *Clarity:* Is the paper well written?

Using this information, you may choose not to read further. This could be because the paper doesn't interest you, or you don't know enough about the area to understand the paper, or that the authors make invalid assumptions. The first pass is adequate for papers that aren't in your research area, but may someday prove relevant.

Incidentally, when you write a paper, you can expect most reviewers (and readers) to make only one pass over it. Take care to choose coherent section and sub-section titles and to write concise and comprehensive abstracts. If a reviewer cannot understand the gist after one pass, the paper will likely be rejected; if a reader cannot understand the highlights of the paper after five minutes, the paper will likely never be read.

2.2 The second pass

In the second pass, read the paper with greater care, but ignore details such as proofs. It helps to jot down the key points, or to make comments in the margins, as you read.

1. Look carefully at the figures, diagrams and other illustrations in the paper. Pay special attention to graphs. Are the axes properly labeled? Are results shown with error bars, so that conclusions are statistically significant? Common mistakes like these will separate rushed, shoddy work from the truly excellent.

2. Remember to mark relevant unread references for further reading (this is a good way to learn more about the background of the paper).

Important!

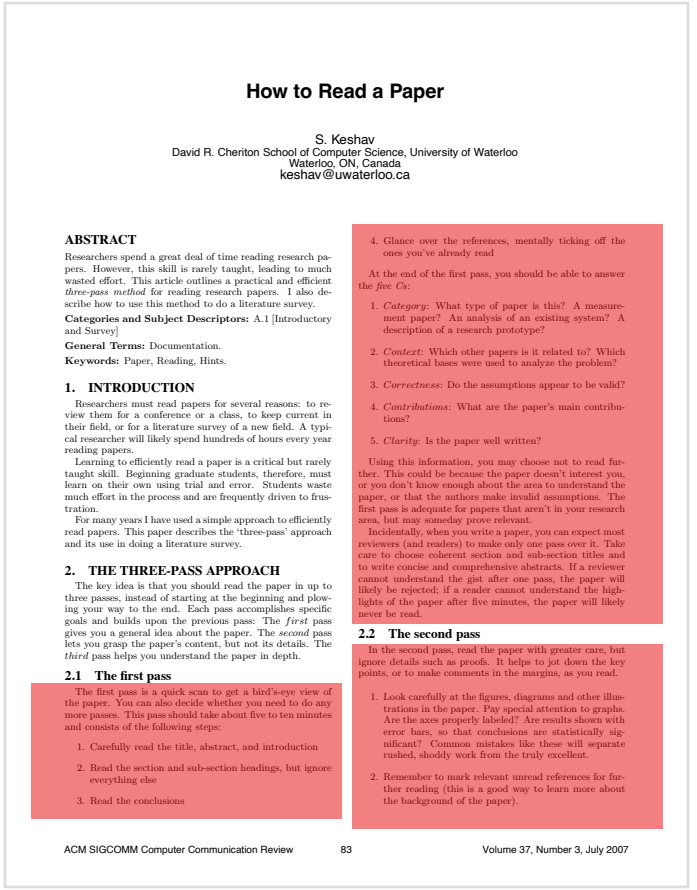
Good to know!

Take notes or jot down points; up to an hour

Third pass

Virtually re-implement the paper

- Appreciate innovations
- Identify shortcomings
- Can take 4-5 hours for beginners
- Up to an hour for experienced readers



How to Read a Paper

S. Keshav
David R. Cheriton School of Computer Science, University of Waterloo
Waterloo, ON, Canada
keshav@uwaterloo.ca

ABSTRACT

Researchers spend a great deal of time reading research papers. However, this skill is rarely taught, leading to much wasted effort. This article outlines a practical and efficient *three-pass method* for reading research papers. I also describe how to use this method to do a literature survey.

Categories and Subject Descriptors: A.1 [Introductory and Survey]

General Terms: Documentation.

Keywords: Paper, Reading, Hints.

1. INTRODUCTION

Researchers must read papers for several reasons: to review them for a conference or a class, to keep current in their field, or for a literature survey of a new field. A typical researcher will likely spend hundreds of hours every year reading papers.

Learning to efficiently read a paper is a critical but rarely taught skill. Beginning graduate students, therefore, must learn on their own using trial and error. Students waste much effort in the process and are frequently driven to frustration.

For many years I have used a simple approach to efficiently read papers. This paper describes the 'three-pass' approach and its use in doing a literature survey.

2. THE THREE-PASS APPROACH

The key idea is that you should read the paper in up to three passes, instead of starting at the beginning and plowing your way to the end. Each pass accomplishes specific goals and builds upon the previous pass: The *first* pass gives you a general idea about the paper. The *second* pass lets you grasp the paper's content, but not its details. The *third* pass helps you understand the paper in depth.

2.1 The first pass

The first pass is a quick scan to get a bird's-eye view of the paper. You can also decide whether you need to do any more passes. This pass should take about five to ten minutes and consists of the following steps:

1. Carefully read the title, abstract, and introduction
2. Read the section and sub-section headings, but ignore everything else
3. Read the conclusions
4. Glance over the references, mentally ticking off the ones you've already read

At the end of the first pass, you should be able to answer the *five Cs*:

1. *Category*: What type of paper is this? A measurement paper? An analysis of an existing system? A description of a research prototype?
2. *Context*: Which other papers is it related to? Which theoretical bases were used to analyze the problem?
3. *Correctness*: Do the assumptions appear to be valid?
4. *Contributions*: What are the paper's main contributions?
5. *Clarity*: Is the paper well written?

Using this information, you may choose not to read further. This could be because the paper doesn't interest you, or you don't know enough about the area to understand the paper, or that the authors make invalid assumptions. The first pass is adequate for papers that aren't in your research area, but may someday prove relevant.

Incidentally, when you write a paper, you can expect most reviewers (and readers) to make only one pass over it. Take care to choose coherent section and sub-section titles and to write concise and comprehensive abstracts. If a reviewer cannot understand the gist after one pass, the paper will likely be rejected; if a reader cannot understand the highlights of the paper after five minutes, the paper will likely never be read.

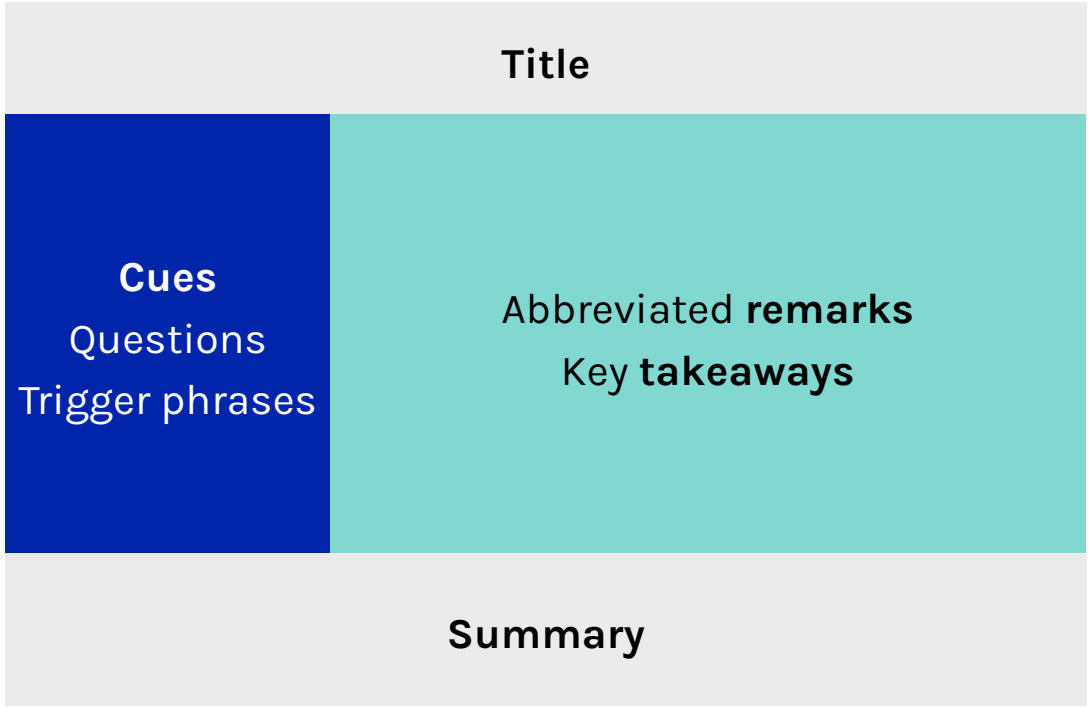
2.2 The second pass

In the second pass, read the paper with greater care, but ignore details such as proofs. It helps to jot down the key points, or to make comments in the margins, as you read.

1. Look carefully at the figures, diagrams and other illustrations in the paper. Pay special attention to graphs. Are the axes properly labeled? Are results shown with error bars, so that conclusions are statistically significant? Common mistakes like these will separate rushed, shoddy work from the truly excellent.
2. Remember to mark relevant unread references for further reading (this is a good way to learn more about the background of the paper).

Taking notes

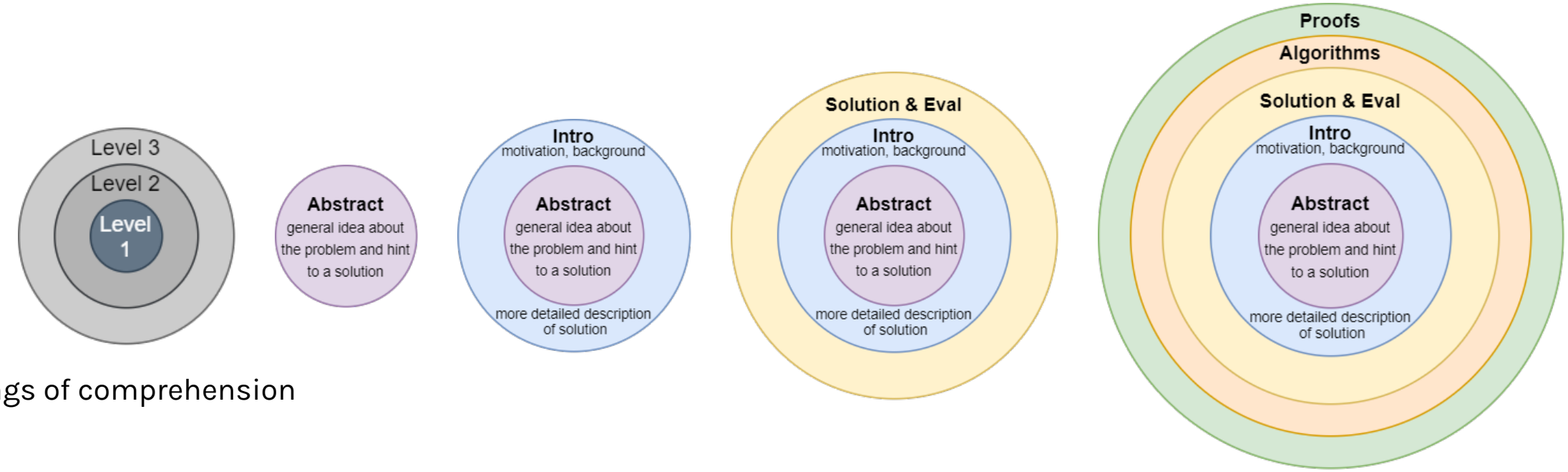
The cornell note-taking system



https://lsc.cornell.edu/how-to-study/taking-notes/cornell-note-taking-system/

The shampoo algorithm

Lather, rinse, and repeat



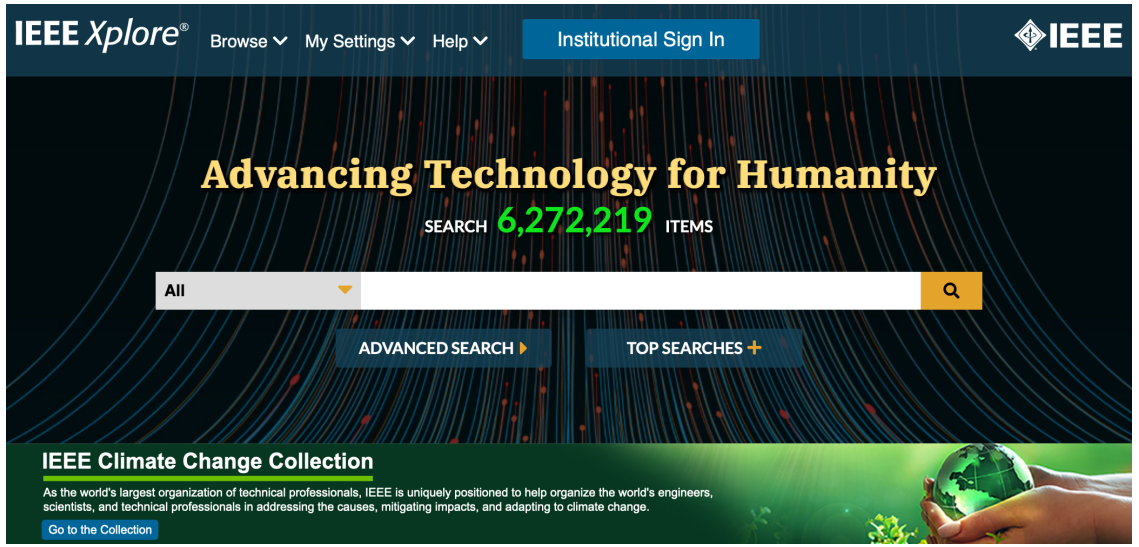
Rings of comprehension

Writing as a reading tool: our brain is too fast when we read; slow it down

https://charap.co/how-to-read-computer-science-systems-papers-using-shampoo-algorithm/

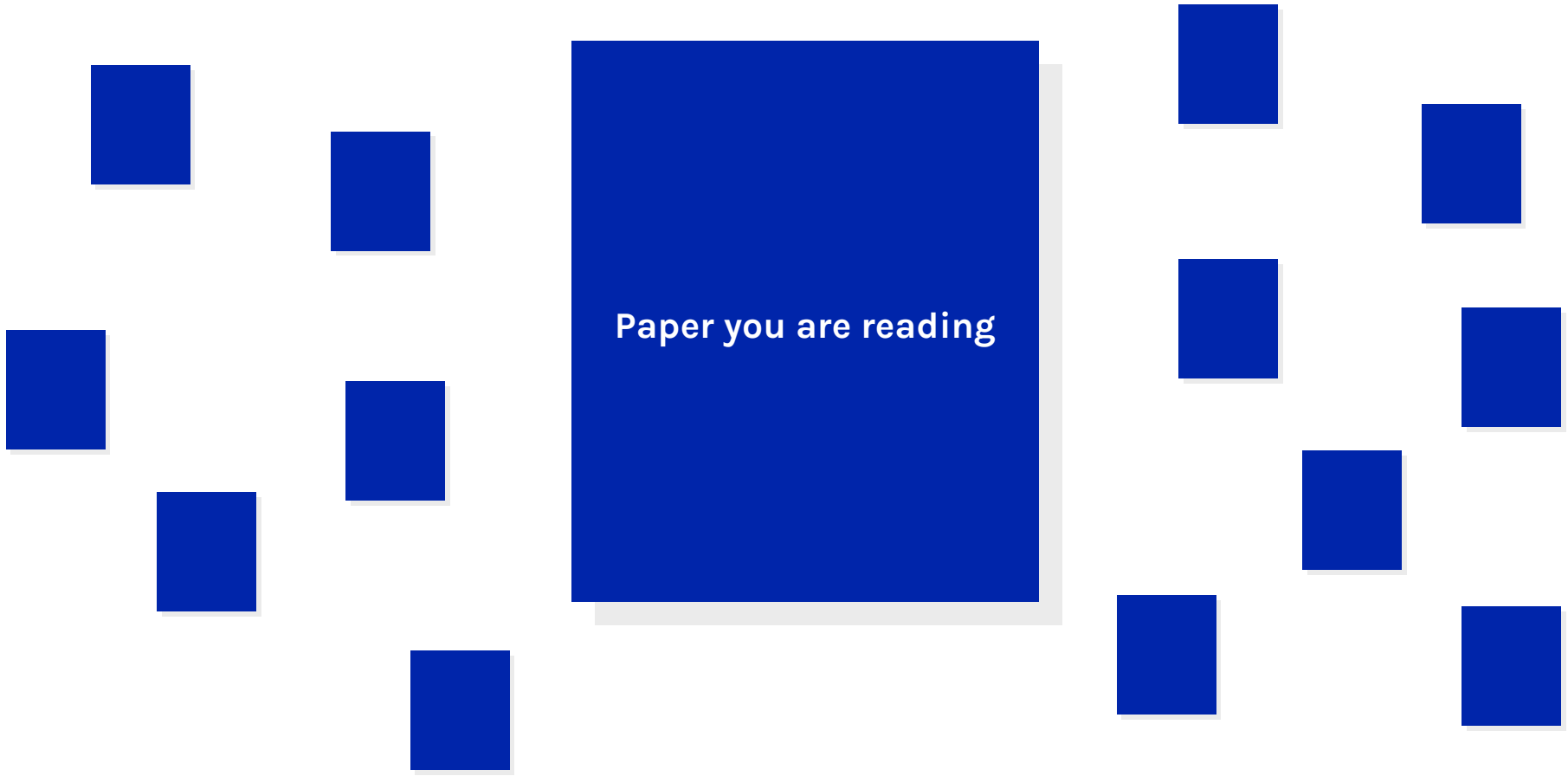
How to find (good) papers to read

Where to download a paper?

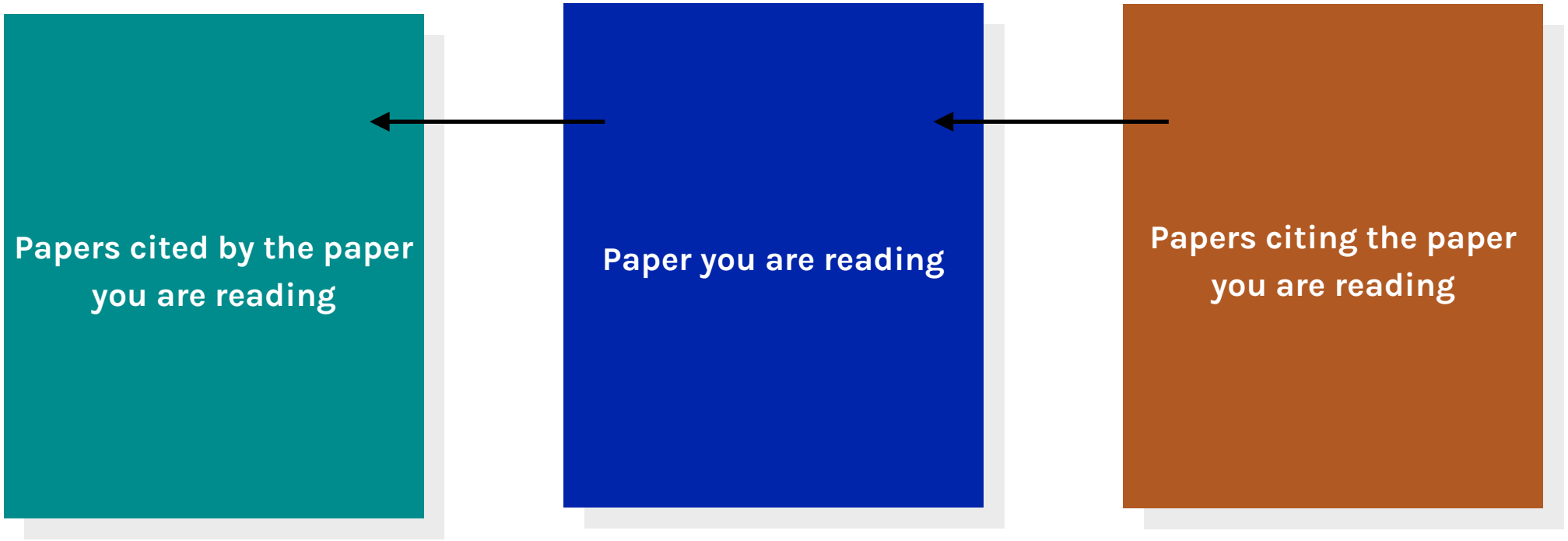


UPB provides subscriptions so you can download papers without a paywall if you use the campus network or VPN

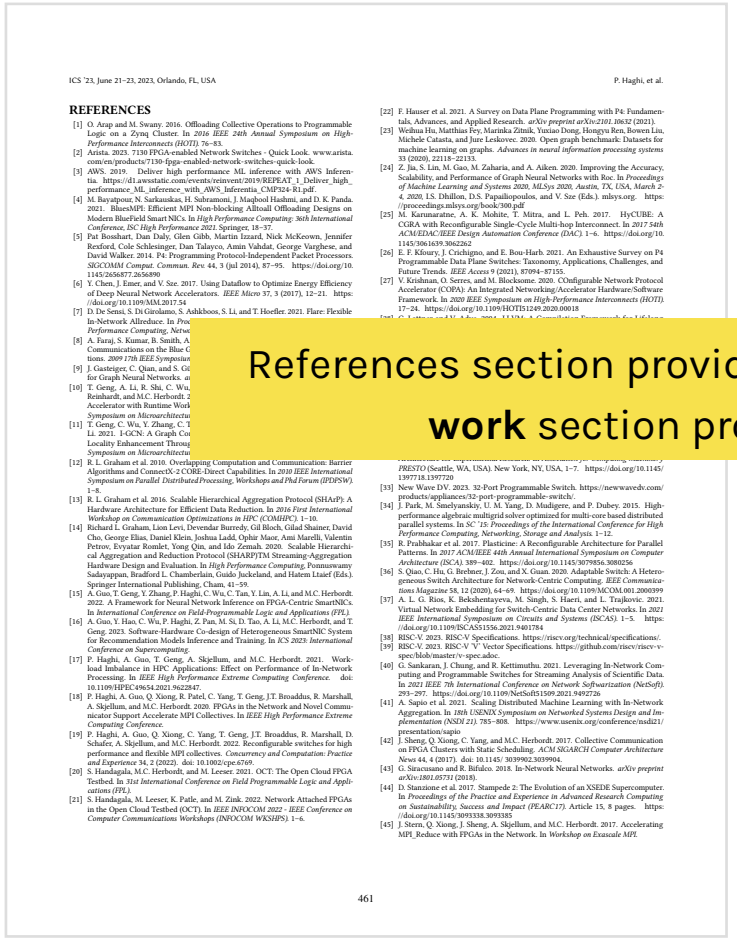
How to find relevant papers?



How to find relevant papers?



Papers cited by the paper you are reading



ICS '23, June 21–23, 2023, Orlando, FL, USA P. Haghi, et al.

REFERENCES

[1] O. Arap and M. Swany. 2016. Offloading Collective Operations to Programmable Logic on a Zynq Cluster. In *2016 IEEE 24th Annual Symposium on High-Performance Interconnects (HOTI)*. 76–83.

[2] Arista. 2023. 7130 FPGA-enabled Network Switches - Quick Look. www.arista.com/en/products/7130-fpga-enabled-network-switches-quick-look.

[3] AWS. 2019. Deliver high performance ML inference with AWS Inferentia. https://d1.awsstatic.com/events/reinvent/2019/REPEAT_1_Deliver_high_performance_ML_inference_with_AWS_Inferentia_CMP324-R1.pdf.

[4] M. Bayatpour, N. Sarkauskas, H. Subramoni, J. Maqbool Hashmi, and D. K. Panda. 2021. BluesMPI: Efficient MPI Non-blocking Alltoall Offloading Designs on Modern BlueField Smart NICs. In *High Performance Computing: 36th International Conference, ISC High Performance 2021*. Springer, 18–37.

[5] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (jul 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[6] Y. Chen, J. Emer, and V. Sze. 2017. Using Dataflow to Optimize Energy Efficiency of Deep Neural Network Accelerators. *IEEE Micro* 37, 3 (2017), 12–21. https://doi.org/10.1109/MM.2017.54

[7] D. De Sensi, S. Di Girolamo, S. Ashkboos, S. Li, and T. Hoefler. 2021. Flare: Flexible In-Network Allreduce. In *Proc... Performance Computing, Netw...*

[8] A. Faraj, S. Kumar, B. Smith, A... Communications on the Blue G... tions. *2009 17th IEEE Symposium...*

[9] J. Gasteiger, C. Qian, and S. Gü... for Graph Neural Networks. a...

[10] T. Geng, A. Li, R. Shi, C. Wu, ... Reinhardt, and M.C. Herbordt. 2... Accelerator with Runtime Work... *Symposium on Microarchitectur...*

[11] T. Geng, C. Wu, Y. Zhang, C. T... Li. 2021. I-GCN: A Graph Co... Locality Enhancement Throug... *Symposium on Microarchitectur...*

[12] R. L. Graham et al. 2010. Overlapping Computation and Communication: Barrier Algorithms and ConnectX-2 CORE-Direct Capabilities. In *2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*. 1–8.

[13] R. L. Graham et al. 2016. Scalable Hierarchical Aggregation Protocol (SHArP): A Hardware Architecture for Efficient Data Reduction. In *2016 First International Workshop on Communication Optimizations in HPC (COMHPC)*. 1–10.

[14] Richard L. Graham, Liuo Levi, Devendar Burredy, Gil Bloch, Gilad Shainer, David Cho, George Elias, Daniel Klein, Joshua Ladd, Ophir Maor, Ami Marelli, Valentin Petrov, Evyatar Romlet, Yong Qin, and Ido Zemah. 2020. Scalable Hierarchical Aggregation and Reduction Protocol (SHARP)TM Streaming-Aggregation Hardware Design and Evaluation. In *High Performance Computing*, Ponnuswamy Sadayappan, Bradford L. Chamberlain, Guido Juckeland, and Hatem Ltaief (Eds.). Springer International Publishing, Cham, 41–59.

[15] A. Guo, T. Geng, Y. Zhang, P. Haghi, C. Wu, C. Tan, Y. Lin, A. Li, and M.C. Herbordt. 2022. A Framework for Neural Network Inference on FPGA-Centric SmartNICs. In *International Conference on Field-Programmable Logic and Applications (FPL)*.

[16] A. Guo, Y. Hao, C. Wu, P. Haghi, Z. Pan, M. Si, D. Tao, A. Li, M.C. Herbordt, and T. Geng. 2023. Software-Hardware Co-design of Heterogeneous SmartNIC System for Recommendation Models Inference and Training. In *ICS 2023: International Conference on Supercomputing*.

[17] P. Haghi, A. Guo, T. Geng, A. Skjellum, and M.C. Herbordt. 2021. Workload Imbalance in HPC Applications: Effect on Performance of In-Network Processing. In *IEEE High Performance Extreme Computing Conference*. doi: 10.1109/HPEC49654.2021.9622847.

[18] P. Haghi, A. Guo, Q. Xiong, R. Patel, C. Yang, T. Geng, J.T. Broaddus, R. Marshall, A. Skjellum, and M.C. Herbordt. 2020. FPGAs in the Network and Novel Communicator Support Accelerate MPI Collectives. In *IEEE High Performance Extreme Computing Conference*.

[19] P. Haghi, A. Guo, Q. Xiong, C. Yang, T. Geng, J.T. Broaddus, R. Marshall, D. Schafer, A. Skjellum, and M.C. Herbordt. 2022. Reconfigurable switches for high performance and flexible MPI collectives. *Concurrency and Computation: Practice and Experience* 34, 2 (2022). doi: 10.1002/cpe.6769.

[20] S. Handagala, M.C. Herbordt, and M. Leeser. 2021. OCT: The Open Cloud FPGA Testbed. In *31st International Conference on Field Programmable Logic and Applications (FPL)*.

[21] S. Handagala, M. Leeser, K. Patle, and M. Zink. 2022. Network Attached FPGAs in the Open Cloud Testbed (OCT). In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 1–6.

[22] F. Hauser et al. 2021. A Survey on Data Plane Programming with P4: Fundamentals, Advances, and Applied Research. *arXiv preprint arXiv:2101.10632* (2021).

[23] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems* 33 (2020), 22118–22133.

[24] Z. Jia, S. Lin, M. Gao, M. Zaharia, and A. Aiken. 2020. Improving the Accuracy, Scalability, and Performance of Graph Neural Networks with Roc. In *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*, I.S. Dhillon, D.S. Papailiopoulos, and V. Sze (Eds.). mlsys.org. https://proceedings.mlsys.org/book/300.pdf

[25] M. Karunaratne, A. K. Mohite, T. Mitra, and L. Peh. 2017. HyCUBE: A CGRA with Reconfigurable Single-Cycle Multi-hop Interconnect. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/3061639.3062262

[26] E. F. Kfoury, J. Crichigno, and E. Bou-Harb. 2021. An Exhaustive Survey on P4 Programmable Data Plane Switches: Taxonomy, Applications, Challenges, and Future Trends. *IEEE Access* 9 (2021), 87094–87155.

[27] V. Krishnan, O. Serres, and M. Blocksome. 2020. COnfigurable Network Protocol Accelerator (COPA): An Integrated Networking/Accelerator Hardware/Software Framework. In *2020 IEEE Symposium on High-Performance Interconnects (HOTI)*. 17–24. https://doi.org/10.1109/HOTI51249.2020.00018

[28] [illegible]

[29] [illegible]

[30] [illegible] *PRESTO* (Seattle, WA, USA). New York, NY, USA, 1–7. https://doi.org/10.1145/1397718.1397720

[33] New Wave DV. 2023. 32-Port Programmable Switch. https://newwavedv.com/products/appliances/32-port-programmable-switch/.

[34] J. Park, M. Smelyanskiy, U. M. Yang, D. Mudigere, and P. Dubey. 2015. High-performance algebraic multigrid solver optimized for multi-core based distributed parallel systems. In *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[35] R. Prabhakar et al. 2017. Plasticine: A Reconfigurable Architecture for Parallel Patterns. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 389–402. https://doi.org/10.1145/3079856.3080256

[36] S. Qiao, C. Hu, G. Brebner, J. Zou, and X. Guan. 2020. Adaptable Switch: A Heterogeneous Switch Architecture for Network-Centric Computing. *IEEE Communications Magazine* 58, 12 (2020), 64–69. https://doi.org/10.1109/MCOM.001.2000399

[37] A. L. G. Rios, K. Bekshentayeva, M. Singh, S. Haeri, and L. Trajkovic. 2021. Virtual Network Embedding for Switch-Centric Data Center Networks. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5. https://doi.org/10.1109/ISCAS51556.2021.9401784

[38] RISC-V. 2023. RISC-V Specifications. https://riscv.org/technical/specifications/.

[39] RISC-V. 2023. RISC-V 'V' Vector Specifications. https://github.com/riscv/riscv-v-spec/blob/master/v-spec.adoc.

[40] G. Sankaran, J. Chung, and R. Kettimuthu. 2021. Leveraging In-Network Computing and Programmable Switches for Streaming Analysis of Scientific Data. In *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*. 293–297. https://doi.org/10.1109/NetSoft51509.2021.9492726

[41] A. Sapio et al. 2021. Scaling Distributed Machine Learning with In-Network Aggregation. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 785–808. https://www.usenix.org/conference/nsdi21/presentation/sapio

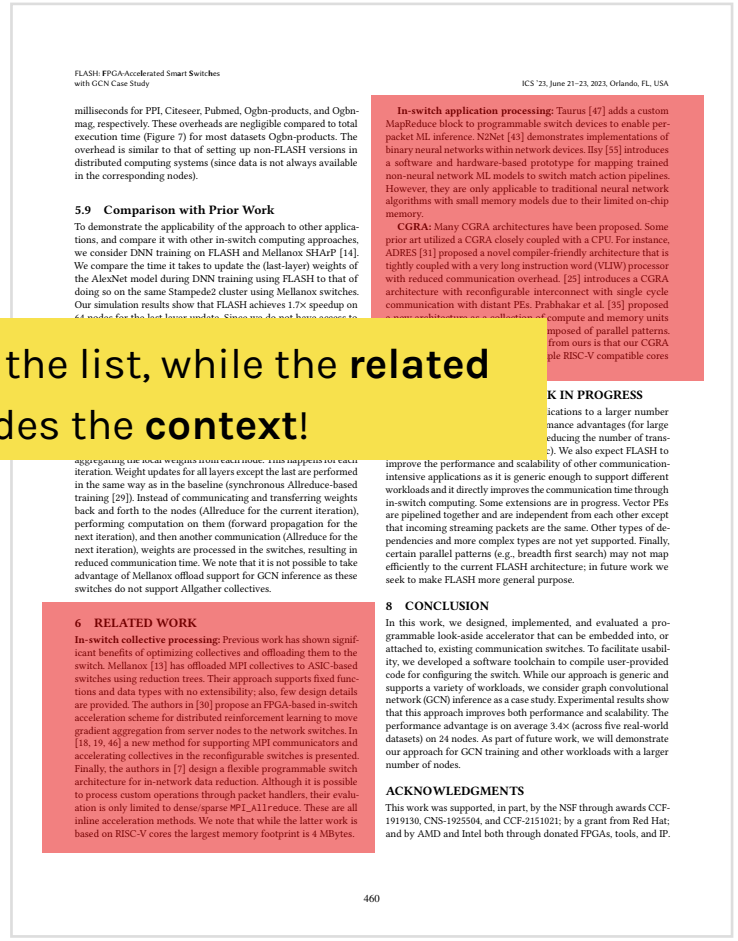
[42] J. Sheng, Q. Xiong, C. Yang, and M.C. Herbordt. 2017. Collective Communication on FPGA Clusters with Static Scheduling. *ACM SIGARCH Computer Architecture News* 44, 4 (2017). doi: 10.1145/ 3039902.3039904.

[43] G. Siracusano and R. Bifulco. 2018. In-Network Neural Networks. *arXiv preprint arXiv:1801.05731* (2018).

[44] D. Stanzione et al. 2017. Stampede 2: The Evolution of an XSEDE Supercomputer. In *Proceedings of the Practice and Experience in Advanced Research Computing on Sustainability, Success and Impact (PEARC17)*. Article 15, 8 pages. https://doi.org/10.1145/3093338.3093385

[45] J. Stern, Q. Xiong, J. Sheng, A. Skjellum, and M.C. Herbordt. 2017. Accelerating MPI_Reduce with FPGAs in the Network. In *Workshop on Exascale MPI*.

References section provides the list, while the **related work** section provides the **context**!



milliseconds for PPI, Citeseer, Pubmed, Ogbn-products, and Ogbn-mag, respectively. These overheads are negligible compared to total execution time (Figure 7) for most datasets Ogbn-products. The overhead is similar to that of setting up non-FLASH versions in distributed computing systems (since data is not always available in the corresponding nodes).

5.9 Comparison with Prior Work

To demonstrate the applicability of the approach to other applications, and compare it with other in-switch computing approaches, we consider DNN training on FLASH and Mellanox SHArP [14]. We compare the time it takes to update the (last-layer) weights of the AlexNet model during DNN training using FLASH to that of doing so on the same Stampede2 cluster using Mellanox switches. Our simulation results show that FLASH achieves 1.7× speedup on [illegible] nodes for the last layer update. Since we do not have access to [illegible] aggregating the local weights from each node. This happens for each iteration. Weight updates for all layers except the last are performed in the same way as in the baseline (synchronous Allreduce-based training [29]). Instead of communicating and transferring weights back and forth to the nodes (Allreduce for the current iteration), performing computation on them (forward propagation for the next iteration), and then another communication (Allreduce for the next iteration), weights are processed in the switches, resulting in reduced communication time. We note that it is not possible to take advantage of Mellanox offload support for GCN inference as these switches do not support Allgather collectives.

6 RELATED WORK

In-switch collective processing: Previous work has shown significant benefits of optimizing collectives and offloading them to the switch. Mellanox [13] has offloaded MPI collectives to ASIC-based switches using reduction trees. Their approach supports fixed functions and data types with no extensibility; also, few design details are provided. The authors in [30] propose an FPGA-based in-switch acceleration scheme for distributed reinforcement learning to move gradient aggregation from server nodes to the network switches. In [18, 19, 46] a new method for supporting MPI communicators and accelerating collectives in the reconfigurable switches is presented. Finally, the authors in [7] design a flexible programmable switch architecture for in-network data reduction. Although it is possible to process custom operations through packet handlers, their evaluation is only limited to dense/sparse `MPI_Allreduce`. These are all inline acceleration methods. We note that while the latter work is based on RISC-V cores the largest memory footprint is 4 MBytes.

In-switch application processing: Taurus [47] adds a custom MapReduce block to programmable switch devices to enable per-packet ML inference. N2Net [43] demonstrates implementations of binary neural networks within network devices. Ilsy [55] introduces a software and hardware-based prototype for mapping trained non-neural network ML models to switch match action pipelines. However, they are only applicable to traditional neural network algorithms with small memory models due to their limited on-chip memory.

CGRA: Many CGRA architectures have been proposed. Some prior art utilized a CGRA closely coupled with a CPU. For instance, ADRES [31] proposed a novel compiler-friendly architecture that is tightly coupled with a very long instruction word (VLIW) processor with reduced communication overhead. [25] introduces a CGRA architecture with reconfigurable interconnect with single cycle communication with distant PEs. Prabhakar et al. [35] proposed [illegible] composed of parallel patterns. [illegible] from ours is that our CGRA [illegible] ple RISC-V compatible cores

K IN PROGRESS

[illegible] ications to a larger number [illegible] mance advantages (for large [illegible] educing the number of trans-[illegible]). We also expect FLASH to improve the performance and scalability of other communication-intensive applications as it is generic enough to support different workloads and it directly improves the communication time through in-switch computing. Some extensions are in progress. Vector PEs are pipelined together and are independent from each other except that incoming streaming packets are the same. Other types of dependencies and more complex types are not yet supported. Finally, certain parallel patterns (e.g., breadth first search) may not map efficiently to the current FLASH architecture; in future work we seek to make FLASH more general purpose.

8 CONCLUSION

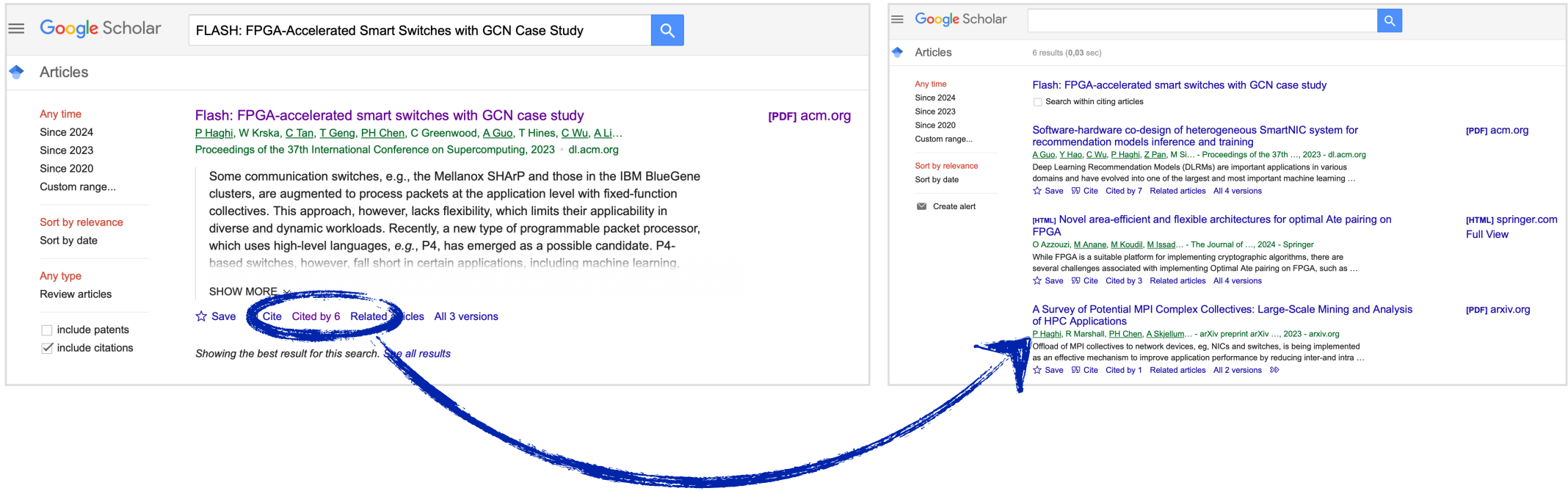
In this work, we designed, implemented, and evaluated a programmable look-aside accelerator that can be embedded into, or attached to, existing communication switches. To facilitate usability, we developed a software toolchain to compile user-provided code for configuring the switch. While our approach is generic and supports a variety of workloads, we consider graph convolutional network (GCN) inference as a case study. Experimental results show that this approach improves both performance and scalability. The performance advantage is on average 3.4× (across five real-world datasets) on 24 nodes. As part of future work, we will demonstrate our approach for GCN training and other workloads with a larger number of nodes.

ACKNOWLEDGMENTS

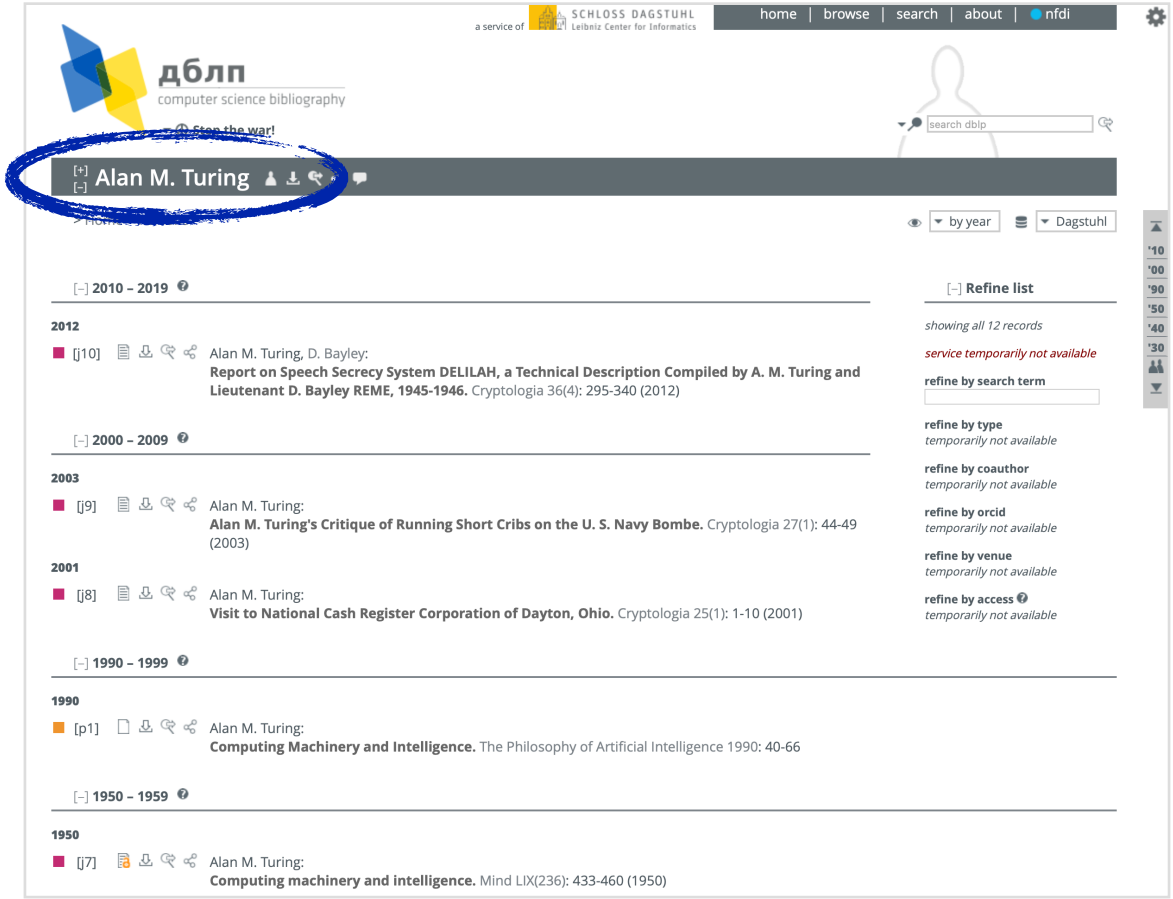
This work was supported, in part, by the NSF through awards CCF-1919130, CNS-1925504, and CCF-2151021; by a grant from Red Hat; and by AMD and Intel both through donated FPGAs, tools, and IP.

Papers citing the paper you are reading

Usually this means more recent papers…



How to find all papers from a particular author?



https://dblp.org/pid/t/AlanMTuring.html

How to discuss a paper

Discussing a paper: peel the onion

Top level

- What is the cool idea?
- Why does it matter?
- How, when, where, and why does it work?

What questions do we have about the paper?

Discussing a paper: peel the onion

Mid level

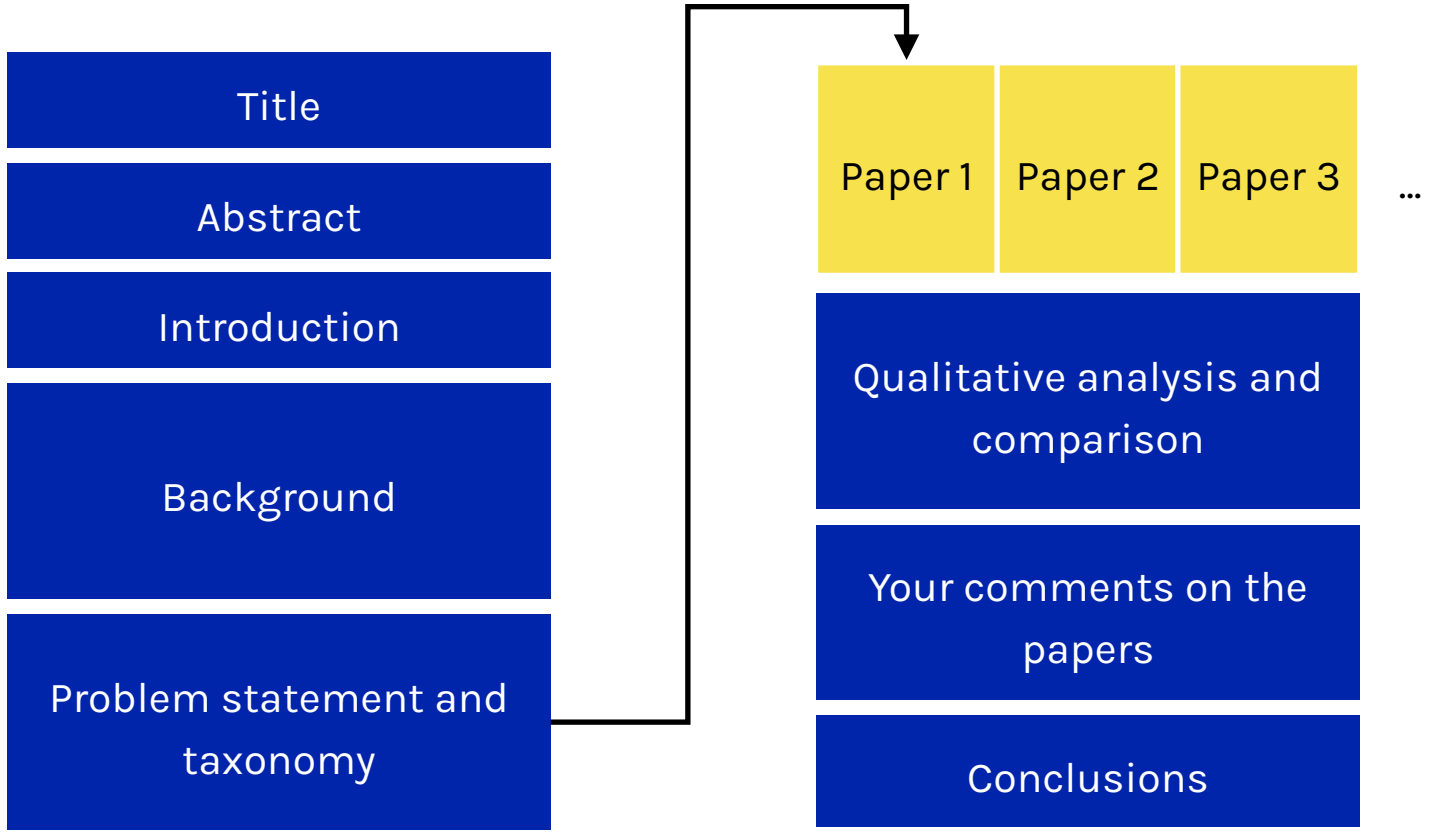
- What are the assumptions? How is the work scoped?
- What is the evaluation?
  - Setup, workload, choice of experiments
- Does the evaluation support the claims?
- Does the paper do a good job with related work (including existing systems)?
- Did you enjoy reading the paper? Are you excited about the ideas?

Discussing a paper: peel the onion

The heart of the discussion

- How does the work advance the state-of-the-art?
- What are the paper's strengths and limitations?
- Will it have a big impact? If so, how?
  - For older papers, does it stand the test of time?
- How does this paper stack up against reality?
  - Is the work applicable in the real world?
  - Do other systems solve the same problem differently?

Basic structure of your report



PADERBORN UNIVERSITY

Title of the Document in One Line

Author of the Document

1 Section Heading

1.1 Subsection Heading

1.1.1 Subsubsection Heading (Avoid Using It If Possible)

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

The above shows a normal paragraph for this document. By default, the paragraph is not indented. If you want to cite a reference, you can use the `\cite` command. Here is an example: HIRE is a novel resource scheduler for in-network computing [1]. The list of references is shown at the end of the document in the "References"

A template will be provided: 10-20 pages.

- First item
- Second item
- Third item
- Last item
  - First subitem
  - Second subitem

1. First entry
2. Second entry
3. Third entry
   a. First subentry
   b. Second subentry

If you have some text you want to put in monospace (e.g., cite something in verbatim), you can use the `\verb` command to do that. Alternatively, you can use `\mintinline{...}{...}`. The difference is that the latter is highlighted with a light gray background and we can also turn on syntax highlighting for many programming or scripting languages. Here is an example to compare these two: `exit 0` and `exit 0`. For this reason, the latter is always preferred when it comes to code.

If you want to write a code block, you can use the `minted` environment, where you can turn on the syntax highlighting if you want. Here is an example for a shell script.

```
echo "Hello world!"
```

The following is an example for a C code snippet.

```
int main(int argc, char** argv) {
  return 0;
}
```

Questions?