

DECLARATION OF NATHANIEL E FRANK-WHITE

1. I am a Records Request Processor at the Internet Archive. I make this declaration of my own personal knowledge.
2. The Internet Archive is a website that provides access to a digital library of Internet sites and other cultural artifacts in digital form. Like a paper library, we provide free access to researchers, historians, scholars, and the general public. The Internet Archive has partnered with and receives support from various institutions, including the Library of Congress.
3. The Internet Archive has created a service known as the Wayback Machine. The Wayback Machine makes it possible to browse more than 450 billion pages stored in the Internet Archive's web archive. Visitors to the Wayback Machine can search archives by URL (i.e., a website address). If archived records for a URL are available, the visitor will be presented with a display of available dates. The visitor may select one of those dates, and begin browsing an archived version of the Web. Links on archived files in the Wayback Machine point to other archived files (whether HTML pages or other file types), if any are found for the URL indicated by a given link. For instance, the Wayback Machine is designed such that when a visitor clicks on a hyperlink on an archived page that points to another URL, the visitor will be served the archived file found for the hyperlink's URL with the closest available date to the initial file containing the hyperlink.
4. The archived data made viewable and browseable by the Wayback Machine is obtained by use of web archiving software that automatically stores copies of files available via the Internet, each file preserved as it existed at a particular point in time.
5. The Internet Archive assigns a URL on its site to the archived files in the format `http://web.archive.org/web/[Year in yyyy][Month in mm][Day in dd][Time code in hh:mm:ss]/[Archived URL]` aka an "extended URL". Thus, the extended URL `http://web.archive.org/web/19970126045828/http://www.archive.org/` would be the URL for the record of the Internet Archive home page HTML file (`http://www.archive.org/`) archived on January 26, 1997 at 4:58 a.m. and 28 seconds (1997/01/26 at 04:58:28). The date indicated by an extended URL applies to a preserved instance of a file for a given URL, but not necessarily to any other files linked therein. Thus, in the case of a page constituted by a primary HTML file and other separate files (e.g., files with images, audio, multimedia, design elements, or other embedded content) linked within that primary HTML file, the primary HTML file and the other files will each have their own respective extended URLs and may not have been archived on the same dates.
6. Attached hereto as Exhibit A are true and accurate copies of screenshots of the Internet Archive's records of the archived files for the URLs and the dates specified in the attached coversheet of each printout.

7. Attached hereto as Exhibit B are true and accurate copies of the Internet Archive's records of the archived files for the URLs and the dates specified in the attached coversheet of each file.
8. I declare under penalty of perjury that the foregoing is true and correct.

DATE: November 30, 2022

Nathaniel E Frank-White
Nathaniel E Frank-White

EXHIBIT A

<https://web.archive.org/web/20050401023020/http://osu.okstate.edu/>

OSU OKLAHOMA STATE UNIVERSITY STILLWATER

TOUR EVENTS APPLY

Academics | Athletics | Research | Administration | News

Future Students
Undergraduate
Graduate
Current Students
Tuition Information
Faculty & Staff
Alumni & Visitors
The OSU System
OSU Extension
OSU Outreach

STUDENT **SIS**
INFORMATION
SYSTEM

Strategic Planning
Campus Master Plan
E-Mail Search
Campus Map
Information Technology
Library Online
Giving to OSU
Employment
Hire OSU Grads
Stillwater Weather
Contact OSU
Site Map
Help



Quick Links

Go

Spotlight

U.S. News & World Report Ranks OSU Medical School, Occupational Education Program Among Nation's Best

Oklahoma State University fared well in the latest U.S. News & World Report rankings, with its medical school and Occupational Education Studies program...

OSU-DAC Partnership Helping Solve Global Pollution Problem Caused by Munitions Salvage and TNT Production

An Oklahoma State University research team working with the United States Army Defense Ammunition Center (DAC) in McAlester, Okla. has developed a functional prototype that uses only sunlight and a proprietary catalyst to detoxify

water ...
more...

Campus Events

Thursday, March 31, 2005

Students w/ 65+ hours enroll
OSU-STW

Friday, April 01, 2005

Students w/60+ hours enroll
OSU-STW

Saturday, April 02, 2005

AFRICAN NIGHT 2005
OSU-STW SERETEAN CENTER

Monday, April 04, 2005

Students w/ 50+ hours enroll
OSU-STW

Please visit our [Events](#) page for other upcoming events.

Google

Search:

Search

okstate.edu

Web

OSU-System | OSU-Stillwater | OSU-OKC | OSU-Tulsa | OSU-Okmulgee | OSU-CHS
Accessibility | Privacy Policy | Copyright

Oklahoma State University - Stillwater | Stillwater, OK 74078 | 405.744.5000
Contact Webmaster | Copyright © 2004 | Oklahoma State University | All rights reserved



<https://web.archive.org/web/20050403020708/http://osu.okstate.edu/academics/>


[Schedule
A Tour](#)

[Apply
To
OSU](#)
[Skip Navigation](#)

[OSU-Stillwater Home](#)
[Future Students](#)
[Undergraduate](#)
[Graduate](#)
[Current Students](#)
[Tuition Information](#)
[Faculty & Staff](#)
[Alumni & Visitors](#)
[The OSU System](#)
[OSU Extension](#)
[OSU Outreach](#)

SIS
 STUDENT
 INFORMATION
 SYSTEM

[Strategic Planning](#)
[Campus Master Plan](#)
[E-Mail Search](#)
[Campus Map](#)
[Information Technology](#)
[Library Online](#)
[Giving to OSU](#)
[Employment](#)
[Hire OSU Grads](#)
[Stillwater Weather](#)
[Contact OSU](#)
[Site Map](#)
[Help](#)

[OSU-Stillwater Home](#) || [Academics](#)

Academics

[Administration](#)
[Centers](#)
[Colleges](#)

[Library](#)
[Services](#)
[Special Programs](#)

ADMINISTRATION

[Division of Academic Affairs](#)
[Faculty SISWEB](#)
[Multicultural Affairs](#)
[Registrar](#)
[Scholarship & Financial Aid](#)
[Student SISWEB](#)
[Undergraduate Admissions](#)
[University Assessment](#)

CENTERS

[Center for Poets and Writers](#)
[COE Technology Resource Center \(COE TRC\)](#)
[Ethics Center](#)
[International Education and Outreach](#)
[Multicultural Development and Assessment Center](#)
[NASA Aerospace Education Services Program](#)
[National Clearinghouse of Rehabilitation Training Materials](#)
[Pragmatism Archive](#)
[Psychological Services Center](#)

COLLEGES

[Agricultural Sciences and Natural Resources](#)
[Arts & Sciences](#)
[CEAT Student Services](#)
[Center for Health Sciences](#)
[Center for Veterinary Health Sciences](#)
[Education](#)
[Engineering, Architecture and Technology](#)
[Graduate College](#)
[Honors College](#)
[Human Environmental Sciences](#)
[William S. Spears School of Business](#)

LIBRARY

[Oklahoma State University Library](#)
[Oklahoma State University Library Catalog](#)

SERVICES

[Center for Science Literacy](#)
[Math Learning Resources Center](#)
[Writing Center](#)

SPECIAL PROGRAMS

[Distance Learning Degree Programs](#)
[Independent Study](#)
[Institute for Telecommunications](#)
[K-12 Distance Learning Academy](#)
[School of International Studies](#)
[Student Academic Mentor Program](#)
[Student Disability Services](#)
[Title IX Compliance](#)
[Women's Studies Program](#)

[OSU-System](#) | [OSU-Stillwater](#) | [OSU-OKC](#) | [OSU-Tulsa](#) | [OSU-Okmulgee](#) | [OSU-CHS](#)
[Accessibility](#) | [Privacy Policy](#) | [Copyright](#)

Oklahoma State University - Stillwater | Stillwater, OK 74078 | 405.744.5000
[Contact Webmaster](#) | Copyright © 2004 | Oklahoma State University | All rights reserved



<https://web.archive.org/web/20050403171652/http://www.cas.okstate.edu/>



DEAN'S OFFICE

ALUMNI

OUTREACH

FACULTY LINKS

STUDENT LINKS

CAREER SERVICES

DEGREE PROGRAMS

News & Notes

- [New Dean Selected for College of Arts & Sciences](#)
- [Chemistry's Berlin wins Medal for Teaching Excellence](#)
- [Use our new Online Academic Advising Appointment system!](#)

COLLEGE OF ARTS & SCIENCES

Twenty-four departments and schools provide students choices to meet a wide variety of interests with over 300 faculty members who are committed to both teaching and research.



A&S Academic Departments

A&S Courses

<https://web.archive.org/web/20050403213144/http://cas.okstate.edu/services/degree/majors/index.html>

COLLEGE OF ARTS & SCIENCES

2004-2005 Degree Programs

For 2002-2003 degree sheets, [click here](#).

Department (links to departmental home pages)	Degree Specializations	Degree Type (links to current year degree sheet)	Graduate Degrees
College of Arts and Sciences	American Studies Biomedical Science - (For students admitted early to medical program with 90 credit hours, who transfer one year of medical school back to OSU for degree credit)	BA BS	
Art	Art History Graphic Design Studio Art	BA BFA BA, BFA	
Biochemistry and Molecular Biology	Biochemistry	BS	MS, PhD
Botany	Botany Biotechnology Ecology	BS	MS, PhD
Chemistry	General ACS Approved	BS BS	MS, PhD
Communication Sciences and Disorders	Communication Sciences and Disorders	BS	MA
Computer Science	Computer Science	BS	MS, PhD
Economics	Economics and International Economic Relations	BA, BS	MS, PhD
English	English Creative Writing Film Technical Writing	BA BA BA BA	MA, PhD
Foreign Languages & Literatures	French German Russian Spanish	BA BA BA BA	
Geography	Geography Applied Resource Management	BA, BS BS	MS, PhD
Geology	Geology	BS	MS
History	History	BA	MA, PhD
Journalism/Broadcasting	Advertising and Sales Broadcast Journalism News-Editorial Public Relations	BA, BS	MS
Mathematics	Mathematics	BA, BS	MS, PhD
Microbiology and Molecular Genetics	Cell and Molecular Biology Microbiology Medical Technology	BS BS BS	MS, PhD
Music	Music Elective Studies in Business Performance Education-Instrumental/Vocal	BA BM BM BM	
Philosophy	Philosophy	BA	MA
Physics	Traditional Applied	BS BS	MS, PhD
Political Science	Int'l Relations/Compar. Politics Public Law Public Policy Fire & Emergency Management Admin.	BA, BS	MA MS FEMA
Psychology	Psychology	BA, BS	MS, PhD
Sociology	Sociology Applied Sociology · Corrections/Juvenile Treatment · Pop/Environ/Resource Alloc. · Social Gerontology · Social Services	BA, BS BA, BS BA, BS	MS, PhD
Statistics	Statistics	BS	PhD
Theater	Performance Design/Technology	BA	MA
Zoology	Zoology Biological Science, General Physiology Wildlife/Fisheries Ecology · Communications · Fisheries Mgmt/Rsch · Wildlife Mgmt/Rsch	BS BS BS BS	MS, PhD

<https://web.archive.org/web/20050404010354/http://a.cs.okstate.edu/welcome.html>



Computer Science Department

Oklahoma State University

Welcome to the Computer Science Department at [Oklahoma State University](#) in [Stillwater](#), Oklahoma.

The Computer Science Department is located in the [College of Arts and Sciences](#). The program emphasizes the software or programming aspects of computing, with research directed toward applied computing. The department offers [Bachelors](#), [Masters](#), and [Doctorate](#) degrees. Here are the recent [departmental statistics](#) on number of majors and number of graduates for all of our programs.

The department not only delivers three degree programs on campus in Stillwater, but it also has a Bachelor's and a Master's program in Tulsa at the OSU-Tulsa campus. A number of the Tulsa courses are taught by full-time faculty in Tulsa. Other computer science courses are offered by compressed video either from Stillwater or from Tulsa. Some of these courses may be offered to companies located in Oklahoma at the same time.

The Computer Science Department **does NOT OFFER Distance Learning courses on the Internet.**

The department has [seminars \(postscript version\)](#) of interest at various times during the semester. These include both internal speakers and external speakers. The seminar series is organized and managed by the Public Relations Committee with assistance of the local chapter of the [Association for Computing Machinery](#).

For More Information

- Undergraduate program: Stillwater campus - [Chuck Ellis \(echuck@cs.okstate.edu\)](#)
- Undergraduate program: Tulsa campus - Terri Blevins ([blevint@osu-tulsa.okstate.edu](#))
- Graduate programs: [gradinfo@cs.okstate.edu](#)
- Web site: [www@cs.okstate.edu](#)

Related Sites

- [OSU Catalog](#)
- [OSU Graduate College](#)
- [Campus Map of Building Locations](#)
- [OSU Career Services Office](#)

[Computer Science Department](#)

<https://web.archive.org/web/20050404001543/http://www.cs.okstate.edu/>

Computer Science Department



The Computer Science Department is located in the College of Arts and Sciences. The program emphasizes the software or programming aspects of computing, with research directed toward applied computing. The department offers Bachelors, Masters, and Doctorate degrees.

We have programs in Stillwater and also in Tulsa, Oklahoma at the OSU-Tulsa campus. Most of the courses that are offered in Tulsa are taught in the evening to accommodate those who work.

Computer Science Department
219 Mathematics, Statistics, and Computer Science
Stillwater, OK 74078-1053
Phone: (405) 744-5668
Fax: (405) 744-9097

Free Software



[OSU Computer Science non-java scripted Web Page](#)

[Association for Computing Machinery Web Page \(ACM\)](#)

Copyright © 2004 OSU Computer Science Department:
[Contact Webmaster](#)

<https://web.archive.org/web/20050312003208/http://www.cs.okstate.edu/indexn2.html>



March 11, 2004

Welcome to the Computer Science Department at Oklahoma State University in Stillwater, Oklahoma.



The Computer Science Department is located in the College of Arts and Sciences. The program emphasizes the software or programming aspects of computing, with research directed toward applied computing. The department offers Bachelors, Masters, and Doctorate degrees. We have programs in Stillwater and also in Tulsa, Oklahoma at the OSU-Tulsa campus. Most of the courses that are offered in Tulsa are taught in the evening to accommodate those who work.

The Computer Science Department does **NOT OFFER Distance Learning** courses on the Internet.

NOTE: Microsoft is making an academic alliance with Universities and is allowing students to obtain free software. Obtain free software and information on [MSDN Academic Alliance](#).

- ▷ [Welcome](#)
- ▷ [Semester Information](#)
- ▷ [Departmental Statistics](#)
- ▷ [Departmental Awards](#)
- ▷ [Departmental Seminars](#)
- ▷ [Job Openings \(both Student and Faculty/Staff\)](#)
- ▷ [Tulsa Programs](#)
- ▷ [Faculty and Staff](#)
- ▷ [CS Programs](#)
- ▷ [Computing Resources](#)
- ▷ [New Department Computer Users](#)
- ▷ [Departmental Research Projects](#)
- ▷ [Departmental Technical Reports](#)
- ▷ [Course Descriptions/Syllabi](#)
- ▷ [ACM](#)

Computer Science Department
219 Mathematical Sciences
Stillwater, OK 74078-1053
Phone: (405) 744-5668
Fax: (405) 744-9097

<https://web.archive.org/web/20050208030559/http://www.cs.okstate.edu/facstaff/index.html>



Faculty

Ajith Abraham, Assistant Professor, Ph.D. Monash University, Australia. Major interests: computational intelligence, hybrid soft computing, neural networks, fuzzy systems, evolutionary computation, natural computation techniques, optimization, modern heuristics, scheduling, data mining. Other interests: grid computing, autonomic computing.

Anthony Burrell, Associate Professor, Ph.D., University of Virginia. Major interests: Real time high speed multimedia ATM networks, systems protocol design, performance characteristics, heterogeneous network interconnections, network and traffic management. Other interests: network and distributed operating systems.

John P. Chandler, Professor, Ph.D., Indiana. Major interests: numerical methods, scientific and engineering applications, mathematical and statistical software. Other areas of interest: neural networks, genetic algorithms.

Debao Chen, Visiting Assistant Professor, Ph.D., University of Texas at Austin. Major interests: Approximation Theory, Numerical Analysis.

H. K. Dai Associate Professor, Ph.D, University of Washington. Major interests: Theory of computation, parallel computation. Other interests: computer networks.

K. M. George, Professor, Ph.D., SUNY at Stony Brook. Major interests: parallel and distributed computing. Other areas of interests: computer architecture.

G. E. Hedrick, Regents Service Professor and Head, Ph.D., Iowa State. Major interests: Computational science, engineering applications, compiler implementation, programming languages. Other areas of interest: database applications.

Douglas Heisterkamp, Assistant Professor, Ph.D., University of Nebraska-Lincoln. Major interests: Computer vision and image processing, computer graphics. Other areas of interest: artificial intelligence, software engineering, parallel computing, and real-time systems.

Istvan Jonyer, Assistant Professor, Ph.D., The University of Texas at Arlington. Major interests: Artificial intelligence, machine learning, data mining, graph-based data mining, fraud detection. Other interests: graph algorithms, AI algorithms and applications in networking/internet/web, intelligent environments, pervasive computing.

Blayne E. Mayfield, Associate Professor, Ph.D., Missouri-Rolla. Major interests: artificial intelligence, object-oriented programming. Other areas of interest: neural networks, genetic algorithms.

Marcin Paprzycki, Assistant Professor, Ph.D, Southern Methodist University. Major interests: Parallel computing, distributed systems, scientific computing, numerical linear algebra, performance analysis, mathematical software. Other areas of interest: neural networks, pattern recognition, genetic algorithms.

Nohpill Park, Assistant Professor, Ph.D., Texas A & M. Major interests: Computer architecture, systems testing and fault tolerance, performance evaluation, reliability.

Mansur H. Samadzadeh, Professor, Ph.D., Center for Advanced Computer Studies (CACS), University of Louisiana at Lafayette (ULL), Lafayette, Louisiana [Formerly: University of Southwestern Louisiana (USL)]. Major interests: software engineering (software metrics, re-use, human factors, debugging, object-oriented design and programming), operating systems (multiprocessor scheduling, synchronization, memory management), parallel computing, simulation, formal languages, automata theory, theory of computation.

Venkatesh Sarangan, Assistant Professor, Ph.D. Penn State. Major interests: Internet computing, QOS in networks, network security, wireless and adhoc networks, stochastic processes and queuing theory, graph theory, modeling and analysis, data mining.

Johnson Thomas, Assistant Professor, Ph.D. University of Reading, Reading, England. Major interests: Computer and network security, e-commerce (security, web services, workplace on demand, etc.), GIS (Geographic Information Systems), wireless ad hoc and sensor networks. Other areas of interest: petri nets, networked multimedia, human-computer interface.

Henri Uehara, Visiting Assistant Professor, MBA, University of California, Berkeley: MSEE, Stanford University. Major interests: computer systems and architecture, wireless technology, venture capital. Other areas of interest: renewable energy, wind and solar power technology.

Staff

Chuck Ellis, Coordinator-Undergraduate Program-Stillwater

Terri Blevins, Academic Counselor - OSU-Tulsa

Beau Turner, Supervising Secretary

Marilyn Bergbower, Office Assistant

Russ Smith, Senior Systems Manager

[Terry Wright](#), Systems Manager

08/24/2003

[Computer Science Department](#)

<https://web.archive.org/web/20050208093855/http://www.cs.okstate.edu/~marcin/>

Marcin Paprzycki

I work in the Department of Computer Science at the [Oklahoma State University](http://www.cs.okstate.edu), Tulsa, OK 74109, USA (e-mail: marcin@cs.okstate.edu, phone: +918-594-8545).

NOTE: For family reasons I am spending the Spring 2005 semester in Poland.

[Parallel and Distributed Computing Practices](#)

- [ADVANCES IN COMPUTATION. Theory and Practice](#)
- [IEEE Distributed Systems](#) (an online publication of IEEE CS)
- [Informatica. An International Journal of Computing and Informatics](#)
- [International Journal of Computer Research](#)
- [Buletinul Stiintific al Universitatii din Baia Mare](#)
- [International Journal of Computational Intelligence](#)
- [International Journal of Computer Science & Applications](#)
- [Poznan Studies in the Philosophy of the Sciences and the Humanities](#)

[Gazeta IT](#) (in Polish)

Conferences:

- [IADIS International Conference Applied Computing 2005](#), Algarve, Portugal, February, 2005
- [Special Track on Agents, Interactions, Mobility, and Systems](#) (AIMS), during the [20th ACM Symposium on Applied Computing \(SAC 2005\)](#), Santa Fe, New Mexico, March, 2005
- [2nd Language & Technology Conference](#), Poznan, Poland, April, 2005
- [8th International Conference on Business Information Systems](#) (BIS'2005), Poznan, Poland, April, 2005
- [38th Annual Simulation Symposium](#), San Diego, CA, April, 2005
- [Cluster Computing and Grid 2005](#), Cardiff, UK, May, 2005
- [Special Track on Intelligent Agent Systems: Theory, Design and Implementation](#), during the [18th International FLAIRS Conference](#), Clearwater Beach, FL, May, 2005
- [International Conference on Computational Science and its Applications](#) (ICCSA 2005), Singapore, May, 2005
- [International Conference on Computational Science](#) (ICCS 2005), Atlanta, GA, May, 2005
- [7th International Conference on Enterprise Information Systems](#), Miami, FL, May, 2005
- [The Fourth IEEE International Workshop on Soft Computing as Transdisciplinary Science and Technology](#) (WSTST'05), Muroan, Japan, May, 2005
- [The 2005 High Performance Computing & Simulation \(HPC&S\) Conference](#), Riga, Latvia, June, 2005
- [Information and Communication Technologies International Symposium](#) (ICTIS'2005), Tetuan, Morocco, June, 2005
- [Sixth International Conference on Communications in Computing](#) (CIC), Las Vegas, NV, June, 2005
- [6th International Conference on Algorithms and Architectures for Parallel Processing](#) (ICA3PP-2005), Melbourne, Australia, June, 2005
- [Workshop on Web and Grid Services for Scientific Data Analysis](#) (WAGSSDA), Oslo, Norway, June, 2005
- [4th International Symposium on Parallel and Distributed Computing](#) (ISPDC), Lille, France, July, 2005
- [Fourth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks](#) (HeteroPar'05), Boston, Massachusetts, USA, September, 2005
- [5th International Conference on Intelligent Systems Design and Applications](#) (ISDA'2005), Wroclaw, Poland, September, 2005
- [Sixth International Conference on Parallel Processing and Applied Mathematics](#) (PPAM 2005), Poznan, Poland, September, 2005
- [Advanced Concepts for Intelligent Vision Systems](#) (ACIVS'2005), Antwerp, Belgium September, 2005
- [HeteroPar'05](#), Boston, Massachusetts, USA, September, 2005
- [The International Symposium on System Theory, Automation, Robotics, Computers, Informatics, Electronics and Instrumentation](#) (SINTES 12), Craiova, Romania, October, 2005
- [International Conference on Computer & Information Science](#) (ICCS'2005), Kolhapur, India, December, 2005

Recent Activities

- [Agent Systems Laboratory](#)
 - Published papers related to the *E-travel project* can be uploaded [here](#)
- [Intelligent Database Agents for Geospatial Knowledge Integration and Management](#)
- [Proceedings of the Second Southern Conference on Computing](#)
- [Proceedings of the First Southern Symposium on Computing](#)
- For those who find this entertaining my complete CV (the only advantage of looking into it is that it is current - last update 09/25/2004 , while this site is not up to date in many areas):
 - [Postscript](#)
 - [PDF](#)
 - [Word](#)

Research Interests

- [Computing](#)
 - [Some projects with my students](#)
 - [Some projects by my students](#)
- [Education](#)
- [Philosophy](#)
- Courses taught this semester
 - [Computing for Scientists and Engineers](#) (CS 4113), WEB BASED
 - [Social Issues in Computing](#) (CS 4883), WEB BASED
- [Courses taught in the past](#)

 Bulgarian
Colleagues



- [University of Rousse](#) (very slow link)
- [Central Laboratory for Parallel Processing](#) Bulgarian Academy of Sciences
- [Favorite Wife](#)
- Favorite Daughters:
 - [New Photos \(Summer 2004\)](#)
 - [Old Photos](#)
 - [Very Old Photos](#)
- Very Very Old Photos and Info:
 - [Kalina Anna](#)
 - [Anna Magdalena](#)
- [Favorite University](#)
- [Favorite Search Engine](#) (it was Yahoo!, but I changed my mind)

- Favorites... in the past I worked for:
 - [University of Southern Mississippi](#)
 - and
 - [University of Texas of the Permian Basin](#)

 Privately

- [Inconsequential Experiments With Useless Words](#)
- [A Touch of Humor](#)

[CS Departmental Pages](#)

last updated: 12/05/2004 (more updates soon)

 [error-file:tidyout.log](#)

<https://web.archive.org/web/20050206001843/http://www.cs.okstate.edu/~pdcp/>

[Reload Home Page](#)

[error-file:tidyout.log](#)

[About PDCP](#)

[error-file:tidyout.log](#) [Instructions for contributors](#)

[Editorial Board](#)

[error-file:tidyout.log](#) [Editors' research interests](#)

[Calls for papers](#)

[error-file:tidyout.log](#) Call for [overview papers](#)

[error-file:tidyout.log](#) Call for [book reviews](#)

[error-file:tidyout.log](#) Call for [software reviews](#)

[All published volumes](#)

[error-file:tidyout.log](#) [Special issues](#)

[error-file:tidyout.log](#) [Overview papers](#)

[error-file:tidyout.log](#) [Editorials](#)

[error-file:tidyout.log](#) [Book reviews on the web](#)

[error-file:tidyout.log](#) [Software reviews on the web](#)

ISSN 1097-2803

[home page](#)

[about the journal](#)

[editorial board](#)

[calls for papers](#)

[all volumes](#)

[current issue](#)

Volume 4, Number 4, Volume 5, Numbers 1 and 2 have been already published
the table of contents and abstracts of papers will be posted as soon as I will be able to find time to do this
I expect them to be "in" by the end of December
please accept my apologies
Marcin Paprzycki

To obtain a sample copy, please, contact the publisher

[Send](#)

[comments/questions](#)

[stat4u](#)

[Nova Science Books and Journals](#)

400 Oser Avenue

Suite 1600

Hauppauge, NY 11788-3619

Tel: (631) 231-7269

Fax: (631) 231-8175

novascience@earthlink.net

https://web.archive.org/web/20050212184635fw_/http://www.cs.okstate.edu/~pdc/vols/allvols.html

All Volumes:

 Volume 4 (2001)	Number 1 Abstracts	Number 2 Abstracts	Number 3 Abstracts	
 Volume 3 (2000)	Number 1 Abstracts	Number 2 Abstracts	Number 3 Abstracts	Number 4 Abstracts
 Volume 2 (1999)	Number 1 Abstracts	Number 2 Abstracts	Number 3 Abstracts	Number 4 Abstracts
 Volume 1 (1998)	Number 1 Abstracts	Number 2 Abstracts	Number 3 Abstracts	Number 4 Abstracts

https://web.archive.org/web/20050317054429fw_/http://www.cs.okstate.edu/~pdc/vols/vol03/vol03no2.html

Special Issue:
Unconventional Parallel Architectures
Guest Editors: Yakov Fet and Peter Kacsuk

[Editorial: Tool Support for Distributed Real-Time Software](#)

J. Zalewski

[Introduction](#)

Kakov Fet and Peter Kacsuk

SPECIAL ISSUE PAPERS

[Reaction-Diffusion and Excitable Processors: A Sense of the Unconventional](#)

A. Adamatzky

[A Scalable Mutiprocessor for Real-Time Signal Processing Applications](#)

H. Eberle and D. Scherrer

[Conciliating Connectionism and Parallel Digital Hardware](#)

B. Girau

[KPROC - An Instruction Systolic Architecture for Parallel Prefix Applications](#)

B. Schmidt and M. Schimmler

[The Effect of Interword Connectivity in Associative Processing](#)

C. Konstantinopoulos, A. Svolos, D. Serpanos and D. Maritsas

RESEARCH PAPERS

[Design, Implementation and Evaluation of a Parallel Object-Oriented Language A-NETL](#)

T. Baba, T. Yoshinaga, Y. Iwamoto and K. Ootsu

[Book Reviews](#)

https://web.archive.org/web/20050317054433fw_/http://www.cs.okstate.edu/~pdc/vols/vol03/vol03no2abs.html#schmidt

Reaction-Diffusion and Excitable Processors: A Sense of the Unconventional

A. Adamatzky

 Non-linear chemical and excitable media exhibit a wide range of space-time dynamic: from well known circular waves to self-localized excitations. If we take a resting medium and change concentration of reagents or other parameters then diffusive or phase waves are generated and spread all over the medium. The waves interact one with another and form either dissipative structure or a precipitate. All micro-volumes of the medium update their states (local concentrations of reagents) in parallel. Thus, the medium can be thought of as a massive parallel processor, where data and results of a computation are represented by concentration profiles of the reagents. A theory of reaction-diffusion processors is still under development. In the paper we give an account of our personal experience in design of reaction-diffusion and excitable processors, mathematical models and working prototypes. Various aspects of reaction-diffusion computing are illustrated by actual examples of parallel solutions of various problems from computational geometry, optimization on graphs and communication networks, control of mobile robots and implementation of logical operations. Prospective material base for fabrication of reaction-diffusion and excitable processors is also tackled.

A Scalable Multiprocessor for Real-Time Signal Processing Applications

H. Eberle and D. Scherrer

 We present a scalable multiprocessor for real-time signal processing applications. The basic building block is a node with eight processors. Nodes are connected by the switch-based network Switcherland [2]. Innovation highlights are: (a) a network with guaranteed transmission rates and bounded forwarding latencies, (b) a multiprocessor node based on RISC CPUs/DSPs and a communication infrastructure optimized for processing continuous data streams, and (c) an object-oriented programming model familiar from general-purpose systems. These properties make the described system a versatile platform for a wide range of signal processing applications.

Conciliating Connectionism and Parallel Digital Hardware

B. Girau

 The distributed structure of artificial neural networks makes them stand as models of parallel computation. Their very fine grain parallelism uses many information exchanges, so that hardware implementations are more likely to fit neural computations. But the number of operators and the complex connection graph of most usual neural models can not be directly handled by digital hardware devices. Therefore a theoretical and practical framework has been defined to reconcile simple hardware topologies with complex neural architectures. This framework has been designed mainly to meet the demands of configurable digital hardware. *Field programmable neural arrays* (FPNA) are based on an original paradigm of neural computation, so that they compute complex neural functions despite their simplified architectures. This paper focuses on the parallel aspects of the FPNA computation paradigm, from its definition to its applied implementation on FPGAs. FPNAs attest that a connectionist paradigm may represent an actually practical model of parallel computing.

KPROC - An Instruction Systolic Architecture for Parallel Prefix Applications

B. Schmidt and M. Schimmler

 The KPROC (KiloPROCessor) architecture is the first implementation of a parallel computer with 1024 floating-point processors on a single chip. It strictly follows the concept of an instruction systolic array. The modular organisation allows for either building large arrays of many KPROC chips or speeding up small machines with a single KPROC as a coprocessor. This paper presents concept of this parallel computer model as well as the architectural details of the processor design. It is shown that this computer model allows for efficient implementation of parallel prefix computations. A large variety of applications from different areas is presented to demonstrate how parallel prefix computations can be used as key operations for deriving efficient implementations on the KPROC.

The Effect of Interword Connectivity in Associative Processing

C.G. Konstantopoulos, A.E. Svolos, D.N. Serpanos and D.G. Maritsas

 Technology advances enable the design of powerful associative processors. Conventional approaches have resulted in architectures, which can store multiple numbers (fields) in a single memory word and can perform arithmetic operations within each word in parallel. Currently, interword communication for parallel processing is performed over a simple interconnect such as a star network with the comparand register as the hub. We follow an alternative approach and we focus on improved interword connectivity rather than highly sophisticated memory word structures. We introduce an associative processor with a single-stage hypercube interconnection among memory words, and with the word-network interface providing routing function and small buffering (two field-size memory buffers, one for transmission and one for reception). This leads to an efficient, low-cost system that allows parallel data transfers and achieves dramatic performance improvements for numerical applications, as we demonstrate introducing efficient algorithms for matrix multiplication and for 1-D and 2-D FFT, achieving linear and logarithmic complexity, respectively. In addition to the hypercube, we analyze the performance effect of additional interword connectivity: unidirectional and bidirectional shifting between adjacent words. We also demonstrate that the improved interword connectivity results in several efficient primitive operations for numerical applications, such as bit-reversal and perfect-shuffle.

Design, Implementation and Evaluation of a Parallel Object-Oriented Language A-NETL

T. Baba, T. Yoshinaga, Y. Iwamoto and K. Ootsu

 A-NETL is a parallel object-oriented language intended to be used for managing small to massive parallelism with medium grain size. Its design goals are to treat data parallel operations at the same cost as programming languages of the SIMD type, to support various styles of message passing among objects, and to provide several synchronization facilities for realizing autonomous control of objects. Starting with these design principles, we present the syntax and semantics of the language features, and its implementation issues, such as the reduction of message communication cost, efficient implementation of statically and dynamically created massive objects, the realization of synchronization schemes, and the object-to-node allocation scheme to minimize communication cost. We present performance results from the language's implementation on an A-NETL oriented multicomputer, on the AP1000 using the AP1000's message passing library and on a cluster of workstations using the PVM library.

<https://web.archive.org/web/20061106005508/http://www.ntu.edu.sg/publicportal/>



**NANYANG
TECHNOLOGICAL
UNIVERSITY**

Search

in NTU Websites

Go

Quick Links

Please select

- About Us
- Admissions
- Academic Programmes
- Administration
- Events & Notices
- Giving to NTU
- Research Hub

[Prospective Students](#) | [Alumni](#) | [Partners](#) | [Media](#) | [Visitors](#)

[中文版](#)



Adobe Flash Player is no longer supported

Open for Application

- NTU-Waseda Double MBA [Admission in 07]
- The Nanyang MBA [Admission in 07]
- MSc Managerial Economics [Admission - Mar 07]
- Master of Public Administration [Admission - Mar 07]
- MSc (Technopreneurship & Innovation) [Chinese Prog: Admission - Mar 07] [English Prog: Admission - May 07]

[More >>](#)



The NTU Undergraduate Experience

- Promoting the spirit of enterprise - NTU Business Plan Competition 2006
- The Global Immersion Programme
- Undergraduate REsearch on cAMPus

Research Highlights

- The Latest Features from Research Hub
- Top Research News
- NTU's research given Nobel boost

NTU in the News

- Laughter, the best medicine
- Singapore biotech firm strikes gold at Asian innovation awards
- Breakthrough NTU nanophotonic technology to revolutionise information and communications technology industry

ClassACT

A monthly e-zine for alumni & friends

- Top European scientist and Nobel trustee named Provost
- Dons' biomedical startup receives US\$7.5 million in funding

Publications

- @NTU
- Staff Digest

Last Modified : 11/2/2006

[NTU Home](#) | [Contact Us](#)

Best viewed with Microsoft IE 6.0 and Netscape 7.0 and above, Windows CE & Netscape 4.79 not presently supported.
Nanyang Avenue, Singapore 639798. Reg. No. 200604393R Tel: (65) 6791-1744 Fax: (65) 6791-1604.
©2006 Nanyang Technological University. All rights reserved. Copyright, Disclaimer, Privacy Statement.

https://web.archive.org/web/20061006171248/https://wis.ntu.edu.sg/webexe88/owa/wdr_search.index_search



Website Directory Service

Centre for IT Services

Close

- ▶ [Departments/Schools](#)
- ▶ [Clubs/Societies](#)
- ▶ [Halls JCRC](#)
- ▶ [Retail Outlets and Facilities](#)

Click on a letter below to go to the list of Staff/Student name begin with that letter:

- ▶ **Staff** [A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#)
- ▶ **Student** [A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#)

Search (Staff & Student)

Departments / Schools websites :

- ▶ [Alumni Affairs Office](#)
- ▶ [Centre for Chinese Language and Culture](#)
- ▶ [Centre for Continuing Education](#)
- ▶ [Centre for Educational Development](#)
- ▶ [Centre for IT Services](#)
- ▶ [Chinese Heritage Centre](#)
- ▶ [College of Engineering](#)
- ▶ [Corporate Communications Office](#)
- ▶ [Dean of Students' Office](#)
- ▶ [General Studies Programme](#)
- ▶ [Graduate Studies](#)
- ▶ [Innovation and Technology Transfer Office](#)
- ▶ [International Relations Office](#)
- ▶ [International Student Centre](#)
- ▶ [Library](#)
- ▶ [NIE \(National Institute of Education\)](#)
- ▶ [Nanyang Business School](#)
- ▶ [Nanyang Executive Centre](#)
- ▶ [Office of Academic Services](#)
- ▶ [Office of Admissions](#)
- ▶ [Office of Facilities Planning and Management](#)
- ▶ [Office of Finance](#)
- ▶ [Office of Human Resources](#)
- ▶ [Office of Professional Attachments](#)
- ▶ [Office of Research](#)
- ▶ [School of Art, Design and Media](#)
- ▶ [School of Biological Sciences](#)
- ▶ [School of Chemical and Biomolecular Engineering](#)
- ▶ [School of Civil and Environmental Engineering](#)
- ▶ [School of Communication and Information](#)
- ▶ [School of Computer Engineering](#)
- ▶ [School of Electrical and Electronic Engineering](#)
- ▶ [School of Humanities and Social Sciences](#)
- ▶ [School of Materials Science and Engineering](#)
- ▶ [School of Mechanical and Aerospace Engineering](#)
- ▶ [School of Physical and Mathematical Science](#)
- ▶ [Student Affairs Office](#)
- ▶ [Student Counselling Centre](#)

If you have any queries, you can email to [CITS.Helpdesk](mailto:CITS.Helpdesk@ntu.edu.sg).

https://web.archive.org/web/20061006173813/https://wis.ntu.edu.sg/pls/webexe88/wdr_search.index_search?P_category=SF&P_index=B



Website Directory Service

Centre for IT Services

Close

- ▶ [Departments/Schools](#)
- ▶ [Clubs/Societies](#)
- ▶ [Halls JCRC](#)
- ▶ [Retail Outlets and Facilities](#)

Click on a letter below to go to the list of Staff/Student name begin with that letter:

- ▶ **Staff** [A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#)
- ▶ **Student** [A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#)

Search (Staff & Student)

GO

Staff websites begin with letter B :

(If you do/do not want your website to be listed below, please click [here](#))

- ▶ [Badawiyah Binte Hamzah](#)
- ▶ [Baharudin Bin Nordin](#)
- ▶ [Bernhard Schmidt](#)
- ▶ [Bertil Schmidt](#)
- ▶ [Bhattacharyya Saurav](#)
- ▶ [Bjoern Holger Neu](#)
- ▶ [Boey Yin Chiang, Freddy](#)
- ▶ [Bogdan Jaroslaw Falkowski](#)
- ▶ [Boh Wai Fong](#)
- ▶ [Boh-Nah Kiat Joo](#)

If you have any queries, you can email to [CITS Helpdesk](#).

<https://web.archive.org/web/20061009222645/http://www.ntu.edu.sg/home/asbschmidt/>

[Main](#) [Research](#) [Teaching](#) [Publications](#) [Students](#) [CV](#)



Dr Bertil Schmidt

Assistant Professor

School of Computer Engineering
Nanyang Technological University
Singapore

Address: N4-02a-32 Nanyang Avenue
Singapore 639798

Tel: (+65) 6790 5194

Fax: (+65) 6792 6559

E-mail: asbschmidt @ ntu.edu.sg



FYP projects for 2006 available (Please contact me for details)

0 0 2 0 6 4

https://web.archive.org/web/20041026062329fw_/http://www.ntu.edu.sg/home/asbschmidt/paper/JPAA04.htm

Title: A Bit-Serial Floating-Point Unit for a Massively Parallel System on a Chip

File download: 

Abstract:: This paper presents the design of a new bit-serial floating-point unit (FPU). It has been developed for the processors of the Instruction Systolic Array parallel computer model. In contrast to conventional bit-parallel FPUs the bit-serial approach requires a different data format. Our FPU uses an IEEE compliant internal floating-point format that allows a fast least significant bit (LSB)-first arithmetic and can be efficiently implemented in hardware

Reference: M. Schimmmler, B. Schmidt, H.-W. Lang: *A Bit-Serial Floating-Point Unit for a Massively Parallel System on a Chip*, Journal of Parallel Algorithms and Applications, 2004 (to appear).

https://web.archive.org/web/20041026064939fw_/http://www.ntu.edu.sg/home/asbschmidt/paper/PDPTA'03.htm

Title: *Design of a Bit-serial Floating-Point Unit for a Fine-Grained Parallel Processor Array*

File download: 

Abstract: This paper presents the design of a new bit-serial floating-point unit (FPU). It has been developed for the processors of the Instruction Systolic Array parallel computer model. In contrast to conventional bit-parallel FPUs the bit-serial approach requires different data formats. Our FPU uses an IEEE compliant internal floating point format that allows a fast least significant bit (LSB)-first arithmetic and can be efficiently implemented in hardware.

Reference: M. Schimmeler, B. Schmidt, H.-W. Lang, *Design of a Bit-serial Floating-Point Unit for a Fine-Grained Parallel Processor Array*, PDPTA'03, Las Vegas, NV, CSERA, 2003

https://web.archive.org/web/20041026062341fw_/http://www.ntu.edu.sg/home/asbschmidt/paper/PDCP00.htm

Title: *KPROC - An Instruction Systolic Architecture for Parallel Prefix Applications*

File download:  pdf

Abstract: The KPROC architecture is the first implementation of a parallel computer with 1024 floating-point processors on a single chip. It strictly follows the concept of an instruction systolic array. The modular organisation allows for either building large arrays of many KPROC chips or speeding up small machines with a single KPROC as a coprocessor. This paper presents concept of this parallel computer model as well as the architectural details of the processor design. It is shown that this computer model allows for efficient implementation of parallel prefix computations. A large variety of applications from different areas is presented to demonstrate how parallel prefix computations can be used as key operations for deriving efficient implementations on the KPROC.

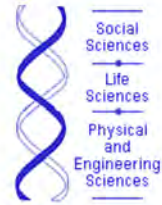
Reference: B. Schmidt, M. Schimmler, *KPROC - An Instruction Systolic Architecture for Parallel Prefix Applications*, Parallel and Distributed Computing Practices, Vol. 3, No. 2, 2000

<https://web.archive.org/web/20011204191802/http://www.nexusworld.com/nova/>

NOVA Science Publishers, Inc.

227 Main Street,
Huntington, NY
11743, USA

Tel: 631-424-6682
Fax: 631-425-5933



E-mail: novascience@earthlink.net

A leading publisher of books and journals with authors from around the world. We publish about 275 books per year and 30 journals. Check out our [company catalog](#) -- we are sure you will find something interesting!

Our site is growing again. In addition to [new book titles](#) every month, we have just added a **NEW** section with a wide range of [journals](#) -- so have a look.



sports computers education cooking
psychology religion biography
poetry health law
KROSHKA & TROITSA on-line book store
books for the discerning reader

[[About the Company](#) | [What's New](#) | [Company Catalog](#) | [Search](#) | [E-mail](#) | [How to Order](#)]



© 1999 NOVA Publishers Inc. All rights reserved.

<https://web.archive.org/web/20011221140039/http://www.nexusworld.com/nova/journals.html>



Journals



CONTENT

- [Current Politics and Economics of the United States](#)
- [Current Politics and Economics of Russia, Eastern and Central Europe](#)
- [Political History and Culture of Russia](#)
- [Current Politics and Economics of Europe](#)
- [Current Politics and Economics of Asia](#)
- [Current Politics and Economics of China](#)
- [International Journal of Energy, Environment and Economics](#)
- [International Journal of Condensed Matter Research](#)
- [International Journal of Theoretical Physics, Group Theory, and Nonlinear Optics](#)
- [Journal of Optics Research](#)
- [Journal of Advanced Materials](#)
- [Journal of Chemical and Biochemical Kinetics](#)
- [International Journal of Exergy](#)
- [International Journal of Commutative Rings](#)
- [Journal of Magnetohydrodynamics, Plasma and Space Research](#)
- [International Journal of Medical and Biological Frontiers](#)
- [Journal of Applied Statistical Science](#)
- [International Journal of Mathematics, Game Theory and Algebra](#)
- [The Internet Journal](#)
- [Parallel and Distributed Computing practices](#)
- [International Journal of Computer Research](#)
- [White House Studies](#)
- [International Journal of Politics and Ethics](#)

<https://web.archive.org/web/20010703022520/http://www.nexusworld.com/nova/jparallel.html>



Parallel and Distributed Computing practices

EDITOR-IN-CHIEF

Marcin Paprzycki

Department of Computer Science and Statistics
University of Southern Mississippi
Hattiesburg, MS 39406-5106, USA
E-mail: pdcp@orca.st.usm.edu
<http://www.orca.st.usm.edu/pdcp>

BOOK REVIEWS EDITOR

Yi Pan

Computer Science Department
University of Dayton
Dayton, OH 45469-2160, USA
pan@cps.udayton.edu

SOFTWARE REVIEWS EDITORS

Hong Shen

School of Computing and Information Technology
Griffith University
Nathan, QLD
4111 Australia
E-mail: hong@cit.gu.edu.au

Domenico Talia

ISI-CNR
c/o DEIS, Università della Calabria
87036 Rende, CS
Italy
talia@si.deis.unical.it

EDITORIAL BOARD

Peter Arbenz, Swiss Federal Inst. of Technology, Zürich, arbenz@inf.ethz.ch
Dorothy Bollman, University of Puerto Rico, bollman@jacobi.upr.clu.edu
Luigi Brugnano, Università di Firenze, brugnano@pontedera.pissoft.it
Bogdan Czejdo, Loyola University, New Orleans, czejdo@beta.loyno.edu
Frederic Desprez, LIP ENS Lyon, Frederic.Desprez@inria.fr
David Du, University of Minnesota, du@cs.umn.edu
Yakov Fet, Novosibirsk Computing Center, fet@ssd.sccc.ru
Len Freeman, University of Manchester, freeman@ma.man.ac.uk
Ian Gladwell, Southern Methodist University, gladwell@seas.smu.edu
John Gustafson, Ames Laboratory, gus@ameslab.gov
Joanna Józefowska, TU, Poznań, joanna@rose.man.poznan.pl
Peter Kacsuk, KFKI-MSZKI Research Institute, kacsuk@sunserv.kfki.hu
David Keyes, Old Dominion University, keyes@cs.odu.edu
Vadim Kotov, Hewlett-Packard Laboratories, kotov@hpl.hp.com
Thomas Ludwig, TU, München, ludwig@informatik.tu-muenchen.de
Veljko Milutinovic, University of Belgrade, emilutiv@etf.bg.ac.yu
Oscar Naim, University of Wisconsin-Madison, naim@cs.wisc.edu
Lalit Patnaik, Indian Institute of Science, lalit@micro.iisc.ernet.in
Viktor Prasanna, University of Southern California, prasanna@usc.edu
Stewart Reddaway, Cambridge Parallel Processing Ltd., sfr@cppuk.co.uk
Siang Wun Song, University of São Paulo, song@ime.usp.br
Bolesław Szymański, Rensselaer Polytechnic Institute, szymansk@cs.rpi.edu
Alejandro Teruel, Universidad Simón Bolívar, teruel@ldc.usb.ve
Kenji Toda, Elechtronical Laboratory, toda@etlca0.etl.go.jp
Daniela Todorova, University of Rousse, daniela@ami.ru.acad.bg
Roman Trobec, Jozef Stefan Institute, roman.trobec@ijs.si
Carl Tropper, McGill University, carl@cs.mcgill.ca
Pavel Tvrđik, Czech Technical University, tvrdik@sun.felk.cvut.cz
Sankait Vahie, i2 Technologies Inc., sam_vahie@i2.com
Jan Van Katwijk, Technical University Delft, J.vanKatwijk@twi.tudelft.nl
Mladen Vouk, North Carolina State University, vouk@adm.csc.ncsu.edu
Lonnie Welch, University of Texas, Arlington, welch@ohio.uta.edu
Janusz Zalewski, University of Central Florida, jza@ece.engr.ucf.edu

Articles of interest

Overview Paper:

Asynchrony in Parallel Computing: From Dataflow to Multithreading
J. Šile, B. Robic and T. Ungerer

Regular Papers: Parallel Computing with Generalized Cellular Automata
W.A. Maniatty, B.K. Szymanski and T. Caraco

A Functional Approach to Radix-r FFTs
D. Bollman, J. Seguel and J. Feo

Visualization of Do-Loop Performance Using AVS
O. Naïm and A.J.G. Hey

Parallel Algorithms for Multidimensional Integration
J.M. Bull and T.L. Freeman

Comparing Parallel Multidimensional Integration Algorithms
I. Gladwell and M.A. Napierała

Calls for Papers and/or Participation

AIMS AND SCOPE

The area of parallel and distributed computing has reached a point where new issues and trends require a professional forum. PDCP will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:

- high level languages
- object oriented techniques
- compiler technology for parallel computing
- implementation techniques and their efficiency

System engineering:

- programming environments
- debugging tools
- software libraries

Performance:

- performance measurement: metrics, evaluation, visualization
- performance improvement: resource allocation and scheduling, I/O, network throughput

Applications:

- database
- control systems
- embedded systems
- fault tolerance
- industrial and business
- real-time
- scientific computing
- visualization

Future:

- limitations of current approaches
- engineering trends and their consequences
- novel parallel architectures

Taking into account the extremely rapid pace of changes in the field PDCP is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. PDCP publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (<http://orca.st.usm.edu/pdcp>). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50-100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in LaTeX using the journal style file (based on the SIAM's *siamltx.sty* style, available at the journal WWW site). Figures must be prepared in PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention (see the journal WWW site for details).

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.

Parameter programming using visual patterns (Vol. 1, #4)

A.S.M. Sajeev

An Application Framework for Dynamic Distributed Software Architectures (Vol. 1, #4)

G. R. Ribeiro-Justo and P. R. F. Cunha

A New Fast Message Passing Communication System for Multiprocessor Workstation Clusters (Vol. 1, #4)

Zheng Weimin, Shen Jun, Wang Ju

Efficient Delay Computation Methods for Real-Time Traffic in an ATM Network (Vol. 1, #4)

Joseph Kee-Yin Ng and Shibin Song

Nova Science Publishers, Inc.

227 Main Street, Suite 100

Huntington , New York 11743

Tel. (631)424-NOVA(6682)

Fax (631)425-5933

Email: Novascience@earthlink.net

<http://www.nexusworld.com/nova>

https://web.archive.org/web/20060924015515fw_/http://www.ntu.edu.sg/home/asbschmidt/publications.htm

Edited Books & Book Chapters

1. B. Schmidt, C. Chen, W. Liu: [Hierarchical Grid Computing for High Performance Bioinformatics](#), in *Grids for Bioinformatics and Computational Biology*, John Wiley & Sons, to appear in 2007
2. B. Schmidt, H. Schröder: [Special-Purpose Computing for Biological Sequence Analysis](#), in *Parallel Computing in Bioinformatics and Computational Biology*, Wiley Book Series on Parallel and Distributed Computing, John Wiley & Sons, New York, 2005 (ISBN: 0-471-71848-3)
3. F.K. Fuss, S. L. Chia, S.S. Venkatraman, S.M. Krishnan, B. Schmidt, [Proceedings of the International BioEngineering Conference 2004 \(IBEC 2004\)](#), Singapore 2004 (ISBN: 981-05-1946-X)
4. B. Schmidt, M. Schimpler, H. Schröder: [High-Speed Cryptography](#), in: *Embedded Cryptographic Hardware: Methodologies & Architectures*, Nova Science Publishers, 2004 (ISBN: 1-59454-012-8)

Journal Papers

1. C. Chen, B. Schmidt, [Constructing Large Suffix Trees on a Computational Grid](#), *Journal of Parallel and Distributed Computing* (accepted for publication)
2. W. Liu, B. Schmidt, [Parallel Pattern-based Systems for Computational Biology: A Case Study](#), *IEEE Transactions on Parallel and Distributed Systems*, Vol. 17, No. 8, pp. 750-763, August 2006.
3. T. Oliver, B. Schmidt, D. Maskell, D. Nathan, R. Clemens, [High-speed multiple sequence alignment on a reconfigurable platform](#), *International Journal of Bioinformatics Research and Applications*, (in press)
4. W. Liu, B. Schmidt, [Mapping of Hierarchical Parallel Genetic Algorithms for Protein Folding onto Computational Grids](#), *IEICE Transactions on Information and Systems*, E89-D(2):589-596, February 2006.
5. M. Schimpler, B. Schmidt, H.-W. Lang, S. Heithecker, [An Area-Efficient Bit-Serial Integer and GF\(2ⁿ\) Multiplier](#), *Microelectronic Engineering*, (in press), DOI: doi:10.1016/j.mee.2006.02.009
6. Z. Du, F. Lin, B. Schmidt, [Accomplishments and Challenges in High Performance Computing for Computational Biology](#), *Current Bioinformatics*, Vol. 1, No. 2, pp. 185-195, May 2006.
7. T. Oliver, B. Schmidt, D. Nathan, R. Clemens, D. Maskell, [Using reconfigurable hardware to accelerate multiple sequence alignment with ClustalW](#), *Bioinformatics*, Vol. 21, No. 16, pp. 3431-3432, August 2005.
8. T. Oliver, B. Schmidt, D. Maskell, [Reconfigurable Architectures for Bio-sequence Database Scanning on FPGAs](#), *IEEE Transactions on Circuits and Systems II*, Vol. 52, No. 12, pp. 851-855, 2005
9. C. Chen, B. Schmidt, [An Adaptive Grid Implementation of DNA Sequence Alignment](#), *Future Generation Computer Systems - The International Journal of Grid Computing: Theory, Methods and Applications*, Vol. 21, No. 7, pp. 988-1003, July 2005.
10. T. Liu, B. Schmidt, [Parallel RNA Secondary Structure Prediction using Stochastic Context-Free Grammars, Concurrency and Computation: Practice and Experience](#), Vol. 17, No. 14, pp. 1669-85, 2005
11. B. Schmidt, M. Schimpler, [A Bit-Serial Floating-Point Unit for a massively parallel system on a chip](#), *Journal of Parallel Algorithms and Architectures*, Vol. 19, No. 2-3, pp. 79-96, 2004.
12. B. Schmidt, M. Schimpler, H. Schröder, [A massively parallel architecture for public-key cryptography](#), *International Journal of Computer Research*, 2004
13. B. Schmidt, F. Lin, A. Laud, Y. Santoso, [Development of Distributed Bioinformatics Applications with GMP, Concurrency and Computation: Practice and Experience](#), Vol. 16, pp. 945-959, 2004.
14. B. Schmidt, H. Schröder, M. Schimpler, [A hybrid architecture for bioinformatics](#), *Future Generation Computer Systems*, No. 18, pp. 855-862, 2002
15. B. Schmidt, H. Schröder, [A hybrid Architecture for Multimedia Processors](#), *Computing and Informatics*, Vol. 20, No. 9, pp. 225-244, 2001
16. B. Schmidt, M. Schimpler, H. Schröder, [Tomographic Image Reconstruction on the Instruction Systolic Array](#), *Computing and Informatics*, Vol. 20, No. 1, pp. 27-42, 2001
17. B. Schmidt, M. Schimpler, [KPROC - An Instruction Systolic Architecture for Parallel Prefix Applications](#), *Parallel and Distributed Computing Practices*, Vol. 3, No. 2, 2000
18. B. Schmidt, H. Schröder, M. Schimpler, [A Morphological Approach to Hough Transform on the Instruction Systolic Array](#), *Computers and Artificial Intelligence*, Vol. 18, No. 6, pp. 541-557, 1999

Conference Papers

1. A. Wirawan, B. Schmidt, [Parallel Discovery of Transcription Factor Binding Sites](#), 2006 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS 2006), Singapore, IEEE Press, to appear.
2. W. Liu, B. Schmidt, G. Voss, W. Müller-Wittig, [GPU-ClustalW: Using Graphics Hardware to Accelerate Multiple Sequence Alignment](#), 13th Annual IEEE International Conference on High Performance Computing (HiPC 2006), Bangalore, India, LNCS (to appear)
3. T. Oliver, B. Schmidt, J. Yanto, D. Maskell, [Accelerating the Viterbi Algorithm for Profile Hidden Markov Models using Reconfigurable Hardware](#), International Conference on Computational Science 2006 (ICCS 2006), Reading, UK, Springer LNCS Vol. 2991, pp. 522-529, 2006.
4. M. Rajapakse, B. Schmidt, V. Brusic, [Multi-Objective Evolutionary Algorithm for Discovering Peptide Binding Motifs](#), in Proc. 4th European Workshop on Evolutionary Computation and Machine Learning in Bioinformatics (EvoBIO 2006), Budapest, Hungary, Springer LNCS 3907, pp. 149-158, 2006.
5. W. Liu, B. Schmidt, G. Voss, A. Schröder, W. Müller-Wittig, [Bio-Sequence Database Scanning on a GPU](#), in Proc. 20th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2006) (HICOMB Workshop), Rhode Island, Greece, 2006.
6. M. Rajapakse, B. Schmidt, V. Brusic, [Pareto-Optimal Motif for Predicting Peptide Binders](#), 16th International Conference on Genomic Informatics (GIW 2005), Yokohama, Japan, 2005
7. G. Voss, A. Schröder, W. Müller-Wittig, B. Schmidt, [Using Graphics Hardware to Accelerate Biological Sequence Analysis](#), *IEEE Tencon 2005*, Melbourne, Australia, 2005
8. W. Liu, B. Schmidt, [Mapping of Hierarchical Parallel Genetic Algorithms for Protein Folding onto Computational Grids](#), *IEEE Tencon 2005*, Melbourne, Australia, 2005.
9. J. Yanto, T. Oliver, B. Schmidt, D. Maskell, [Biological Sequence Analysis with Hidden Markov Models on an FPGA](#), *ACSAC 2005*, Singapore, LNCS 3740, Springer, 2005
10. C. Chen, B. Schmidt, [Parallel Construction of Large Suffix Trees on a PG Cluster](#), *Future Gen. Comp. Syst.*, 2006

10. C. Chen, B. Schmidt, [Parallel Construction of Large Sum-Tree on a PC Cluster](#), **Euro-Par 2005**, Lisbon, Portugal, LNCS 3648, Springer, 2005.
11. M. Rajapakse, L. Wyse, B. Schmidt, V. Brusic: [Deriving Matrix of Peptide-MHC Interactions in Diabetic Mouse by Genetic Algorithm](#), 6th International Conference on Intelligent Data Engineering and Automated Learning (**IDEAL 2005**), Brisbane, Australia, LNCS, Springer, 2005.
12. T. Oliver, B. Schmidt, D. Nathan, R. Clemens, D. Maskell: [Multiple Sequence Alignment on an FPGA](#), 11th International Conference on Parallel and Distributed Systems (**ICPADS 2005**) (Workshops), IEEE Computer Society Press, Fukuoka, Japan, 2005.
13. T. Oliver, B. Schmidt, D. Maskell, A.P. Vinod: [A Reconfigurable Architecture for Scanning Biosequence Databases](#), 2005 IEEE International Symposium on Circuits and Systems (**ISCAS 2005**), Kobe, Japan, IEEE Computer Society Press, 2005.
14. W. Liu, B. Schmidt, [A Case Study on Pattern-based Systems for High Performance Computational Biology](#), 19th IEEE International Parallel & Distributed Processing Symposium (**IPDPS 2005**) (Workshops), Denver, CO, 2005.
15. T. Oliver, B. Schmidt, D. Maskell, [Hyper Customized Processors for Bio-Sequence Database Scanning on FPGAs](#), 13th ACM International Symposium on Field-Programmable Gate Arrays, (**FPGA 2005**), Monterey, CA, 2005.
16. T. Liu, B. Schmidt, [Optimal local alignment in RNA secondary structures](#), IASTED International Conference on Biomedical Engineering (**BioMED 2005**), Innsbruck, Austria, 2005.
17. W. Liu, B. Schmidt: [A Tunable Coarse-Grained Parallel Algorithm for Irregular Dynamic Programming Applications](#), 11th Annual International Conference on High Performance Computing (**HIPC 2004**), Bangalore, India, Springer, LNCS, 2004
18. C.X. Chen, B. Schmidt: [Load Balancing for Hierarchical Grid Computing: A Case Study](#), 11th Annual International Conference on High Performance Computing (**HIPC 2004**), Bangalore, Springer, LNCS, 2004
19. W. Liu, B. Schmidt, [A Generic Parallel Pattern-based System for Bioinformatics](#), **Euro-Par 2004**, Pisa, Italy, LNCS, Springer, 2004
20. C.S.S. Tan, B. Schmidt, [Combining message-passing and inter-process communication in SMP-hybrid cluster for efficient parallel medical image analysis](#), **APPLICATIONS OF DIGITAL IMAGE PROCESSING XXVII**, Denver, CO, Proceedings of SPIE Vol. #5558, 2004
21. T. Oliver, B. Schmidt, [High Performance Biosequence Database Scanning on Reconfigurable Platforms](#), 18th IEEE International Parallel & Distributed Processing Symposium (**IPDPS 2004**) (Workshops), Santa Fe, NM, 2004
22. T. Liu, B. Schmidt, [Parallel RNA Sequence-Structure Alignment](#), 18th IEEE International Parallel & Distributed Processing Symposium (**IPDPS 2004**) (Workshops), Santa Fe, NM, IEEE, 2004
23. C.X. Chen, B. Schmidt, [Performance Analysis of Computational Biology Applications on Hierarchical Grid Systems](#), 4th IEEE/ACM International Symposium on Cluster Computing and the Grid (**CCGrid 2004**), Chicago, IL, IEEE, 2004
24. C.X. Chen, B. Schmidt, [Computing Large-Scale Alignments on a Multi-Cluster](#), IEEE International Conference on Cluster Computing (**Cluster 2003**), Hong Kong, 2003
25. W. Liu, B. Schmidt, [Parallel Design Pattern for Computational Biology and Scientific Computing](#), IEEE International Conference on Cluster Computing (**Cluster 2003**), Hong Kong, 2003
26. M. Schimpler, B. Schmidt, S. Heithecker, [An Area-Efficient Bit-Serial Integer Multiplier](#), The 2003 International Conference on VLSI (**VLSI'03**), Las Vegas, NV, 2003
27. M. Schimpler, B. Schmidt, H.-W. Lang, [Design of a Bit-serial Floating-Point Unit for a Fine-Grained Parallel Processor Array](#), The 2003 International Conference on Parallel and Distributed Processing Techniques and Applications (**PDPTA'03**), Las Vegas, NV, 2003
28. B. Schmidt, F. Lin, A. Laud, Y. Santoso, [Parallel Detection of Regulatory Elements with gMP](#), 17th IEEE International Parallel & Distributed Processing Symposium (**IPDPS 2003**) (Workshops), Nice, France, 2003
29. L. Wang, W. Cai, B. Schmidt, B.S. Lee, W. Jie, [Biogrid Computing Platform: Parallel computing for protein alignment analysis](#), **HPC Asia 2002**, Bangalore, India, 2002
30. H. Schröder, B. Schmidt, J. Zhu, [A new approach to protein structure prediction](#), The International Conference on Bioinformatics (**InCoB 2002**), Bangkok, Thailand, 2002
31. B. Schmidt, M. Schimpler, W. Adi, [Area efficient modular arithmetic for mobile security](#), The 2002 International Conference on Wireless Networks (**ICWN'02**), Las Vegas, NV, 2002
32. B. Schmidt, H. Schröder, M. Schimpler, [Massively Parallel Solutions for Molecular Sequence Analysis](#), 16th IEEE International Parallel & Distributed Processing Symposium (**IPDPS 2002**) (Workshops), Ft Lauderdale, FL, 2002
33. B. Schmidt, H. Schröder, [Massively Parallel Sequence Analysis with Hidden Markov Models](#), International Conference on Scientific & Engineering Computation (**IC-SEC 2002**), Singapore, 2002
34. L. Wang, W. Cai, B. Schmidt, B.S. Lee, W. Jie, [Replica Selection Framework for Bio-Grid Computing](#), International Conference on Scientific & Engineering Computation (**IC-SEC 2002**), Singapore, 2002
35. H. Schröder, B. Schmidt, [Image Processing Algorithms for on-board Computing](#), Third International Conference on Information, Communications & Signal Processing (**ICICS'01**), Singapore, 2001
36. L.S. Chin, H. Schröder, B. Schmidt, [Parallel Volume Rendering using Slice Orientation](#), Third International Conference on Information, Communications & Signal Processing (**ICICS'01**), Singapore 2001
37. B. Schmidt, H. Schröder, M. Schimpler, [Protein Sequence Comparison on the Instruction Systolic Array](#), 6th International Conference on Parallel Computing Technologies (**PaCT'01**), Novosibirsk, Russia, Springer LNCS Vol. 2127, 2001
38. B. Schmidt, H. Schröder, M. Schimpler, [Scanning Biosequence Databases on a Hybrid Parallel Architecture](#), **Euro-Par 2001**, Manchester, UK, Springer LNCS Vol. 2150, 2001
39. B. Schmidt, H. Schröder, T. Srikanthan, [A SIMD Solution to Biosequence Database Scanning](#), Parallel Computing 2001 (**ParCo 2001**), Naples, Italy, Imperial College Press, 2001
40. L.S. Chin, B. Schmidt, H. Schröder, [Volume Rendering Using the Instruction Systolic Array Concept](#), Asia-Pacific Conference on Computational Mechanics, Sydney, Australia, 2001
41. B. Schmidt, [Design of a Parallel Accelerator for Volume Rendering](#), **Euro-Par 2000**, Munich, Germany, Springer LNCS Vol. 1900, pp. 1095-1104, 2000
42. B. Schmidt, H. Schröder, [Volume Visualisation on a Hybrid Parallel Architecture](#), 7th Australasian Conference on Parallel and Real-Time Systems (**PART 2000**), Sydney, Australia, 2000
43. B. Schmidt, M. Schimpler, [A Parallel Accelerator Architecture for Multimedia Video Compression](#), **Euro-Par 1999**, Toulouse, France, Springer LNCS Vol. 1685, 1999
44. B. Schmidt, M. Schimpler, H. Schröder, [Long Operand Arithmetic on Instruction Systolic Computer](#)

- [*Architectures and its Application for RSA Cryptography*](#), **Euro-Par 1998**, Southampton, UK, Springer LNCS Vol. 1470, 1998
45. B. Schmidt, M. Schimmmler, H. Schröder., [*The Instruction Systolic Array in Tomographic Image Reconstruction Applications*](#), 5th Annual Australasian Conference on Parallel and Real-time Systems (**PART 1998**), Adelaide, Australia, 1998
 46. B. Schmidt, M. Schimmmler, H. Schröder, [*Morphological Hough Transform on the Instruction Systolic Array*](#), **Euro-Par 1997**, Passau, Germany, Springer LNCS Vol. 1300, 1997
 47. U. Mahlmeister, B. Schmidt, G. Sommer, [*Preattentive Colour Features by Steerable Filters*](#), 17th DAGM Symposium Pattern Recognition (**DAGM'95**), Bielefeld, Germany, Springer, 1995

EXHIBIT B

https://web.archive.org/web/20070824103510fw_/http://www.ntu.edu.sg/home/asbschmidt/paper/Vegas_277PD.pdf

Design of a Bit-Serial Floating Point Unit for a Fine Grained Parallel Processor Array

Manfred Schimmler¹, Bertil Schmidt² and Hans-Werner Lang³

1 Institut für Datentechnik und Kommunikationsnetze, TU Braunschweig, 38106 Braunschweig, Germany, schimmler@ida.ing.tu-bs.de, Phone: +49 531 3913735, Fax: +49 531 3915296

2 School of Computer Engineering, Nanyang Technological University, Singapore 639798, asbschmidt@ntu.edu.sg, Phone: +65 67905194, Fax: +65 67926559

3 Fachhochschule Flensburg, 24943 Flensburg, Germany, lang@fh-flensburg.de Phone: +49 461 8051235, Fax: +49 461 8051527

Abstract -- This paper presents the design of a new bit-serial floating-point unit (FPU). It has been developed for the processors of the Instruction Systolic Array parallel computer model. In contrast to conventional bit-parallel FPUs the bit-serial approach requires different data formats. Our FPU uses an IEEE compliant internal floating point format that allows a fast least significant bit (LSB)-first arithmetic and can be efficiently implemented in hardware.

Keywords: bit-serial Floating Point Units, massively parallel processors, systolic arrays

I. Introduction

Modern VLSI technology allows to integrate massively parallel systems on a single chip. The area limitations for the processors of such systems require small but efficient computational units. In particular, it may motivate the choice of a bit-serial data organization of the individual processors. The fine-grained processor arrays are used in applications where identical operations have to be performed on large data sets such as image and video processing [8], cryptography [7], biological sequence analysis [9] and scientific computing. Implementations of relevant numerical and signal processing algorithms often require floating arithmetic in order to achieve adequate accuracy. Thus, the integration of efficient floating-point arithmetic is essential to these processors. Motivated by the

design of an Instruction Systolic Array (ISA) with 1024 processors on one chip [6], an FPU has been designed that meets these requirements. Its bit-serial structure allows for a fine grained pipelined implementation on a minimal area of silicon. The design has been guided by experiences with a lower-integrated predecessor version of the processor array, which is available as an add-on board for a PC [4].

This paper is organized as follows. Section 2 describes the ISA principle. An overview of the individual ISA processor architecture is given in Section 3. The design methods for implementing efficient floating-point operations on bit-serial hardware are described in Section 4. Section 5 presents the corresponding execution units. The performance of several floating-point operations is given in Section 6. Section 7 concludes the paper.

II. Principle of the ISA

The ISA [2,3] is a mesh-connected processor grid, where the processors are controlled by three streams of control information: instructions, row selectors, and column selectors (see Fig. 1). The instructions are input in the upper left corner of the processor array, and from there they move step by step in horizontal and vertical direction through the array. This guarantees that within each diagonal of the array the same instruction is active during each clock cycle. In clock cycle $k+1$ processor $(i+1,j)$ and $(i,j+1)$ execute the instruction that has been executed by processor (i,j) in clock cycle k . The selectors also move

systolically through the array: the row selectors horizontally from left to right; the column selectors vertically from top to bottom. Selectors mask the execution of the instructions within the processors, i.e. an instruction is executed if and only if both selector bits, currently in that processor, are equal to one. Otherwise, a no-operation is executed.

Every processor has read and write access to its own memory. Besides that, it has a designated *communication register (C-register)* that can also be read by the four neighbour processors. Within each clock phase reading access is always performed before writing access. Thus, two adjacent processors can exchange data within a single clock cycle in which both processors overwrite the contents of their own C-register with the contents of the C-register of its neighbour. This convention avoids read/write conflicts and also creates the possibility to perform parallel prefix operations efficiently, e.g. row sum.

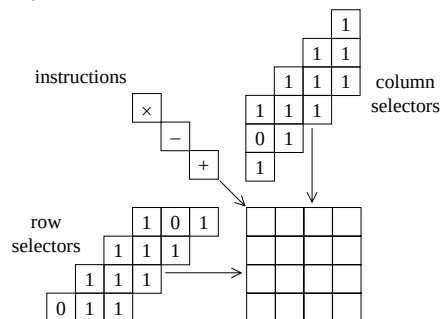


Figure 1: Control flow in an ISA

III. Processor Architecture

The processor architecture is part of a chip design containing an ISA of size 32×32 [6]. The main components of the processor architecture are a set of 64 data registers, the communication register, a unit for integer addition, logical operations and conditional instructions, a 16-bit multiplier, and a shifter/adder for floating point arithmetic (see Fig. 2). The data word length is 16 bits. The data are processed bit-serially, such that the execution of each instruction takes exactly 16 clock cycles. However, there are macro instructions formed as series of elementary instructions in order to handle data types consisting of more than one machine word, e.g. double precision floating point numbers (see Section 4). After receiving an instruction, the

processor stores it in the instruction register, decodes the two operand addresses and the destination address, retrieves the operands from the register file, executes the instruction, writes the result back to the destination register, and passes the instruction to the next processor. Since all this is done bit-serially, it can be pipelined on bit level, such that a new instruction can be fetched and processed every 16 clock cycles. For instance, the first bit of the result is computed already after the second bits of the operands have entered the respective unit.

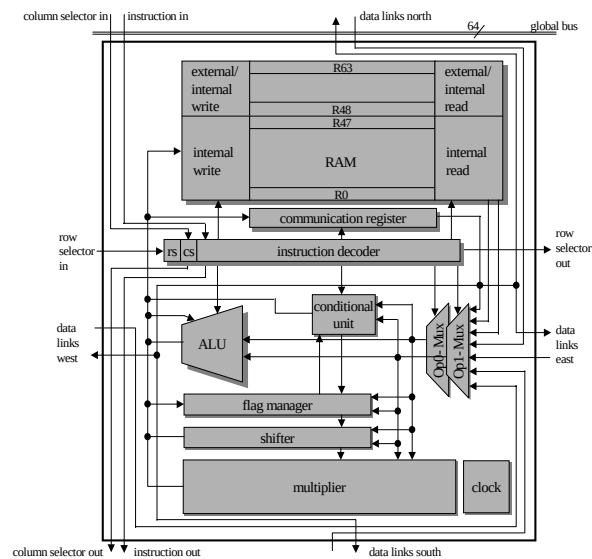


Figure 2: Block diagram of the processor architecture.

The instructions are 3-address instructions (two operand registers *op0* and *op1*; destination register *dest*) coded in two 32 bits. Several instructions are related to floating-point operations. However, these instructions in general do not make sense on their own, but only within a macro instruction, i.e. a sequence of instructions that realize a certain operation on floating point numbers. The instruction

expadd *op0*, *op1*, *dest*

computes the sum of the exponents of two floating point numbers, as required for a multiplication. Similarly, there exist instructions **expsub** for computing the difference of two exponents and **expsel** for selecting the greater exponent as the result exponent in the case of an addition of two floating point numbers. The instructions **shadd** (shift and add), **shsub** (shift and subtract), **nrm** (normalize), **ldmul** (load multiplier), and **mult** (multiply) are used in

macro instructions for computing the mantissa of the result. The instruction

mode length

is used for configuration of the floating point control for handling single precision or double precision numbers. The instructions

mtw op0, dest

expw op0, dest

support the conversion between the IEEE floating point format [1] and the internal format, forming the mantissa words and the exponent word of the internal format.

IV. Floating Point Operations

The processor supports single and double precision floating-point arithmetic. Floating-point registers are implemented by groups of 16-bit data registers, and floating-point operations are implemented as instruction macros, i.e. as sequences of elementary instructions.

A. Internal Floating Point Format

The special cases for IEEE numbers are coded in all bits of the number representation. For example in order to find out if a number equals infinity the complete mantissa and the complete exponent have to be compared with 0 and -1, respectively. Therefore, a bit-serial compare to infinity (required for every arithmetic operation) would need at least 32 internal clock cycles for single and 64 for double. To solve this problem a conversion into an internal format is performed. The internal representation of a double X consists of five 16-bit words $X[0], \dots, X[4]$. Here, $X[4]$ contains the exponent of the double, the sign, the kind of representation (absolute value/sign or 2-complement), and information about possible special cases (infinity, zero, NaN = not a number). $X[0]$ to $X[3]$ contain the significant (mantissa) of the double, extended to 64 bits.

$X[4]$ exp.:

S_m	$T1$	$T0$	R	Se	Se	$e9 \dots e0$
-------	------	------	-----	------	------	---------------

S_m : sign of the significant 0: positive, 1: negative

$T1, T0$: encoding of special cases: 00: normal number, 01: zero, 10: INF (infinity), 11: NaN

R : representation of significant: 0: absolute value/sign representation, 1: 2-complement representation

Se : sign of exponent (two identical bits for overflow detection)

$e0, \dots, e9$: binary representation of the exponent E in 2-complement (little endian mode)

64 bit of the significant $M = (X[3], X[2], X[1], X[0])$, where 63 is the most significant bit:

X[3] significant:	63...48
X[2] significant:	47...32
X[1] significant:	31...16
X[0] significant:	15...0

The value of such a double is:

$0 \cdot (-1)^{S_m}$ if $T0=1$ and $T1=0$;
 $INF \cdot (-1)^{S_m}$ if $T0=0$ and $T1=1$;
 NaN if $T0=1$ and $T1=1$;
 $(-1)^{S_m} \cdot 0.M \cdot 2^E$ if $T0=0$ and $T1=0$ and $R=0$;
 $(-1)^{S_m} \cdot 0.M' \cdot 2^E$ if $T0=0$ and $T1=0$ and $R=1$,
 where M' is the 2-complement of M ;

Double numbers can be normalized or not normalized. A positive number and a number in absolute value/sign-representation is normalized iff bits 63, 62, 61 of the significant are 001, i.e. bits 63 and 62 hold the sign and then in bit 61 comes the most significant bit of the significant. As usual in absolute value/sign representation, the significant is positive and the sign is stored in the S_m -bit of the exponent (see $X[4]$). A negative number in 2-complement is normalized iff bits 63, 62, 61 of the significant are 110. Again, bits 63 and 62 form the sign and it follows the most significant digit of the significant, represented in 2-complement.

The four 16-bit words of the IEEE format are converted into 5 words of the internal format. The 52-bit mantissa is stored in four words, where in the first word an explicit 1 is introduced (this 1 is implicit in the IEEE representation). The first two 0's serve as two sign bits of the mantissa. Two sign bits are useful for detecting overflows: an overflow after an addition changes the second sign bit and a right shift for normalization is necessary. The exponent is stored in the last 12 bits of a separate word, also with two sign bits. The value 01 of the sign bits indicates exponent overflow, whereas 10 indicates exponent underflow. In these cases, the number is changed to INF or 0, respectively. The special values zero, INF, and NaN are coded using the tag stored in bit 14 and 13 of the exponent word. In the case of a special value, the actual value of the mantissa is irrelevant. Bit 12 is used internally for indicating if the mantissa is in 2-complement representation after an addition or subtraction. The conversion is supported by special instructions of the processor.

B. Floating Point Addition

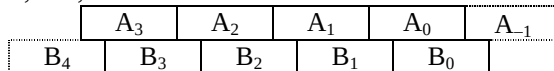
An addition of two floating-point numbers is performed in the usual way: The two exponents are compared, the mantissa of the operand with smaller exponent is shifted to the right appropriately, then the mantissas are added. If any operand is negative and not yet in 2-complement representation, its mantissa is complemented before. The bit-serial organization allows on-the-fly computation with one single half adder. If any operand is 0, its mantissa is set to 0. The result mantissa is in 2-complement representation. The result exponent is the larger exponent of the two operands. The tag of the result is computed depending on the tags of the two operands according to Table 1.

TABLE I. COMPUTATION OF THE RESULT TAGS FOR AN ADDITION

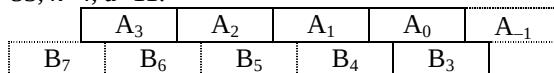
	x	0	INF	NaN
x	$x^{1)}$	x	INF	NaN
0	x	0	INF	NaN
INF	INF	INF	$INF^{2)}$	NaN
NaN	NaN	NaN	NaN	NaN

- 1) except the case that the result mantissa is 0, than the result tag is 0
- 2) except the case that the signs of the two operands are different, then the result tag is NaN

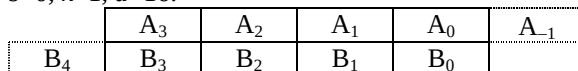
$s=5, k=1, d=11$:



$s=53, k=4, d=11$:



$s=0, k=1, d=16$:



$s \geq 64, k \geq 5$:

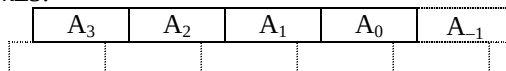


Figure 3: Addition of two 64-bit mantissas.

The sign of the result is the sign of the result mantissa, except in the case that any operand is INF, then the sign is equal to the sign of this operand. If the result is 0 or NaN, the sign is irrelevant. Since the operands consist of more than one word (e.g. of four mantissa words in the case of double precision floating point numbers),

shifting of a mantissa by more than 15 bits is realized by displaced addressing of the operands (see Figure 3).

In each case, in order to realize a shift distance of s , in the first step A_{-1} and B_{-1+k} are addressed, where $k = (s \text{ div } 16) + 1$. B_{-1+k} is loaded into the shifter and delayed by $d = 16 \cdot k - s$ bits. In the next step A_0 and B_{0+k} are addressed. While B_{0+k} is loaded to the shifter, the d bits of B_{-1+k} appear at the output of the shifter and are added to A_0 . In the last step A_3 and B_{3+k} are addressed. However, B_{3+k} is an invalid operand, therefore it is cancelled, i.e. set to 0...0 or to 1...1 depending on the sign of the mantissa. If k becomes greater than 4, all B -operands are cancelled, i.e. set to 0...0. In the case that B has the greater exponent, then the difference of the exponents is negative and A has to be shifted. The situation is essentially the same as above with the roles of A and B interchanged.

C. Normalization

After an addition, in general the result mantissa is not in normalized form, i.e. there are not exactly two leading 0's or 1's. Several cases are possible: there is one leading digit, two leading digits (this is the normalized case), three through 63 leading digits, or 64 leading digits. In the latter case, the result is considered to be 0, if the leading digits are 0. In all other cases a left shift by $-1...62$ is necessary so that the mantissa starts with 001 or 110. The exponent has to be corrected by this shift distance. A counter, the *leading digits counter (LDC)*, is used to count the number of leading digits (minus 2) of the result produced during an addition/subtraction or multiplication. The *LDC* also records the first bit of the mantissa, indicating the sign of the mantissa, and detects if the mantissa is 0. Before a normalization, the value of the *LDC* is transferred into the shift control register. But now this value has to be interpreted in a different way, since it is the distance for a left shift instead of a right shift as in the case of an addition. However, this can be done using the same hardware for normalization as for addition.

D. Floating Point Multiplication

A multiplication of two 64-bit mantissas is performed according to the following scheme

(Fig. 4), where e.g. $A_3 A_2 A_1 A_0 B_0$ denotes the 80-bit product of $A_3 A_2 A_1 A_0$ with B_0 .

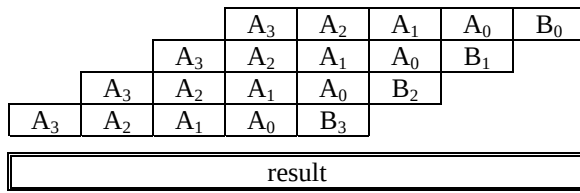


Figure 4: Multiplication scheme for 4-word operands.

The result would be 128 bit long, however, only the most significant 64 bits can be used for the result mantissa F . Therefore, the above scheme is truncated, in order to reduce the number of 16-bit multiplications (Fig. 5):

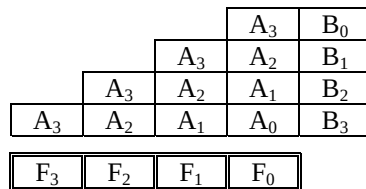


Fig. 5: Truncated multiplication scheme for 4-word operands.

Addition of the intermediate results is performed simultaneously with the multiplications in the multiplier, which is actually a multiply-and-add unit. The error introduced by the truncation amounts to at most 3 bits. This seems to be tolerable, because the precision of the internal floating point format is 61 significant bits, whereas the original IEEE format has only 52 bits. If more precise arithmetic is required, the complete multiplications have to be executed. The exponents of the two floating-point numbers are added. The sign of the result is the exclusive-or of the two-operand signs. The tag of the result is computed according to the following table.

TABLE II. COMPUTATION OF RESULT TAGS FOR A MULTIPLICATION

	x	0	INF	NaN
x	$x^{1)}$	x	INF	NaN
0	x	0	NaN	NaN
INF	INF	NaN	INF	NaN
NaN	NaN	NaN	NaN	NaN

1) except the case of an exponent underflow or overflow, then the result tag is 0 or INF, respectively.

V. Floating Point Units

A. Multiplier

The multiplier can multiply a 16-bit word x with a k -word operand y , yielding a $k+1$ -word result. Simultaneously, the multiplier can add a $k+1$ -word operand s . In the above scheme for floating point multiplication this possibility is used when e.g. $A_3 A_2$ is multiplied with B_1 and added to the result of $A_3 B_0$.

Figure 6 shows the bit-serial 16-bit multiplier used in this architecture. In the first step of a multiplication, the operand x and the first word of operand s have to be loaded into the multiplier (using a `ldmul` instruction). Both are available in bit-serial form with the LSB first. While x is stored temporarily in a register, the $s_0, \dots, s_{15}$ are passed through a sequence of full adders (each adding $s_i + 0$) and delay elements. At the end of the `ldmul` instruction, s_0 is at the input of the rightmost full adder (FA).

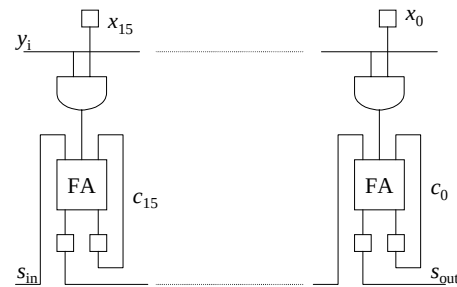


Figure 6: 16-bit multiplier.

In the next instruction cycle, the first word of the other operand y and the next word of s are shifted into the multiplier, while the first word of the result $x \cdot y + s$ is obtained at the output of the multiplier. Again this is done bit-serially: in the first clock cycle, x is multiplied by y_0 and added to s . This step produces bit 0 of the result at s_{out} . The following clock cycles proceed in the same way with $y_1, \dots, y_{15}$. The most interesting property of this multiplier is that it can handle long operands. y is not restricted to 16 bits, but can consist of k 16-bit words, as well as s , which then consists of $k+1$ words. That is, the words of y and s can be input immediately one after the other. In the last instruction cycle, 0's have to be input into the multiplier, while the last word of the result appears at the output.

B. Floating Point Adder

The floating point adder consists of a shifter, an adder, two units for displaced addressing of the operands, two complement units, and a logic for cancellation of operands. The shifter is a configurable shift register of variable length between 0 and 15 bits.

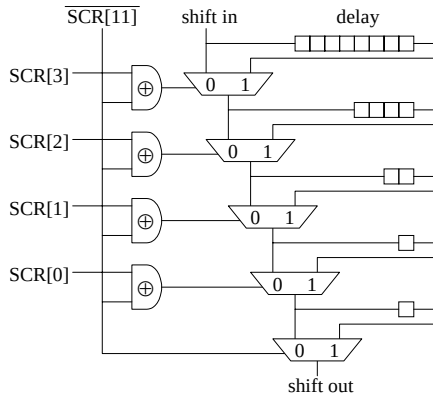


Figure 7: 16-bit shifter.

For a floating-point addition, the result of the exponent comparison is stored in a special register, the shift control register SCR. The highest bit SCR[11] indicates if the exponent difference is negative and determines the shift operand. Bits SCR[0..3] determine the delay of the shifter. If this is a non-negative number s , it is complemented by the XOR-gates and the last 1-bit delay is activated, in order to achieve the required delay of $16-s$. Bits SCR[4..5] are used for displaced addressing of the shift operand, as explained in Section 4. If any of the bits SCR[6..10] is different from SCR[11], the shift distance would be greater than 64, so the shift operand may be cancelled. The shifted operand and the other operand are added by a bit-serial adder. If any operand is negative and not yet in 2-complement representation, it is complemented before the addition.

VI. Performance

For single precision floating point numbers, an addition takes 14 instruction cycles, a multiplication takes 7 instruction cycles. The performance of one processor is

$$\begin{aligned} & 2 \text{ operations} / 21 \text{ instruction cycles} * 1 \\ & \text{instruction cycle} / 16 \text{ clock cycles} * f \text{ clock} \\ & \text{cycles} / \text{second} = f/168 \text{ operations} / \\ & \text{second} \end{aligned}$$

With $f = 200$ MHz we have a performance of one processor of 200/168 million floating point operations/second or 1.2 MFlops; this amounts to 1.2 GFlops for one chip with 1024 processors or to 20 GFlops for a board with 16 chips. For double precision floating point numbers, an addition takes 18 instruction cycles, a multiplication takes 16 instruction cycles. The performance of one processor is

$$\begin{aligned} & 2 \text{ operations} / 34 \text{ instruction cycles} * 1 \\ & \text{instruction cycle} / 16 \text{ clock cycles} * f \text{ clock} \\ & \text{cycles} / \text{second} = f/272 \text{ operations} / \\ & \text{second} \end{aligned}$$

With $f = 200$ MHz we have a performance of one processor of 200/272 million floating point operations/second or 0.735 MFlops; this amounts to 735 MFlops for one chip with 1024 processors or to 12 GFlops for a board with 16 chips.

Also more complex floating point operations can be computed with the presented FPU, e.g. division, square root. Table 3 shows the instruction count for several floating-point operations.

TABLE III. INSTRUCTION COUNT FOR SEVERAL FLOATING POINT OPERATIONS IN SINGLE/DOUBLE PRECISION WITH THE PRESENTED FPU.

Operation	Single	Double
Format Conversion	6	8
Addition/Subtraction	14	18
Multiplication	7	16
Division	52	101
Square root	92	138

VII. Conclusions

In this paper the design of a FPU has been presented. Its bit-serial nature allows for an area efficient and energy efficient implementation. Because bit-serial operations with the IEEE floating point number representation are difficult to implement, we use an internal floating point format that allows a fast LSB-first arithmetic. Our FPU is optimally suited for a fine grained pipelined architecture. That is why it has been used in the processing elements of the KPROC [6], a parallel system with 1024 processors on a single chip.

References

- [1] *IEEE standard for binary floating point arithmetic*, ANSI IEEE754-1985, New York, (1985).
- [2] Kunde, M., et al.: *The Instruction Systolic Array and its Relation to Other Models of Parallel Computers*, Parallel Computing 7, pp. 25-39 (1988)
- [3] Lang, H.-W.: *The Instruction Systolic Array, a parallel architecture for VLSI*, Integration, the VLSI Journal 4, pp. 65-74 (1986)
- [4] Lang, H.-W., Maaß, R., Schimmler, M.: *The Instruction Systolic Array - Implementation of a Low-Cost Parallel Architecture as Add-On Board for Personal Computers*, Proc. HPCN 94, LNCS 797, Springer, pp.487-488 (1994).
- [5] Schimmler, M., Lang, H.-W.: *The Instruction Systolic Array in Image Processing Applications*, Proc. Europto 96, SPIE 2784, pp. 136-144 (1996)
- [6] Schmidt, B., Schimmler, M.: *KPROC – An Instruction Systolic Architecture for Parallel Prefix Applications*, Parallel and Distributed Computing Practices, 3 (2) pp. 163-178 (2000).
- [7] Schmidt, B., Schimmler, M., Schröder, H.: *Long Operand Arithmetic on Instruction Systolic Computer Architectures and Its Application to RSA cryptography*, Euro-Par'98, LNCS 1470, Springer, pp. 916-922 (1998)
- [8] Schmidt, B., Schimmler, M.: *A Parallel Accelerator Architecture for Multimedia Video Compression*, EuroPar'99, LNCS 1685, Springer, pp. 950-960, (1999).
- [9] Schmidt, B., Schröder, H., T. Srikanthan: *A SIMD Solution to Biosequence Database Scanning*, Proc. PARCO'01, pp. 230-237, (2001)

https://web.archive.org/web/20070824102612fw_/http://www.ntu.edu.sg/home/asbschmidt/paper/JPAA04.pdf

A BIT-SERIAL FLOATING-POINT UNIT FOR A MASSIVELY PARALLEL SYSTEM ON A CHIP

MANFRED SCHIMMLER¹, BERTIL SCHMIDT², HANS-WERNER LANG³

¹*Institut für Informatik und Praktische Mathematik, CAU Kiel, 24098 Kiel, Germany,
masch@informatik.uni-kiel.de, Phone: +49 431 880 4480, Fax: +49 431 880 4054*

²*School of Computer Engineering, Nanyang Technological University, N4-02a-32
Nanyang Avenue, Singapore 639798, asbschmidt@ntu.edu.sg, Phone: +65 67905194,
Fax: +65 67926559*

³*Fachhochschule Flensburg, 24943 Flensburg, Germany, lang@fh-flensburg.de Phone:
+49 461 8051235, Fax: +49 461 8051527*

Address for manuscript correspondence:

Bertil Schmidt, School of Computer Engineering, Nanyang Technological University,
N4-02a-32 Nanyang Avenue, Singapore 639798

Email: asbschmidt@ntu.edu.sg, Tel: ++65 67905194, Fax: ++65 67926559

Abstract: This paper presents the design of a new bit-serial floating-point unit (FPU). It has been developed for the processors of the Instruction Systolic Array parallel computer model. In contrast to conventional bit-parallel FPUs the bit-serial approach requires a different data format. Our FPU uses an IEEE compliant internal floating-point format that allows a fast least significant bit (LSB)-first arithmetic and can be efficiently implemented in hardware.

Key Words: bit-serial Floating Point Units, massively parallel processors, systolic arrays;

1 Introduction

Modern VLSI technology allows to integrate massively parallel systems on a single chip. The area limitations for the processors of such systems require small but efficient computational units [11,14,20]. In particular, it may motivate the choice of a bit-serial data organization of the individual processors. The fine-grained processor arrays are used in a variety of applications where identical operations have to be performed on large data sets such as signal, image and video processing [16,20], cryptography [15], biological sequence analysis [17], scientific computing [19], visualization [11,18] and medical imaging [13]. Implementations of relevant numerical and signal processing algorithms often require floating arithmetic in order to achieve adequate accuracy. Thus, the integration of efficient floating-point arithmetic is essential to these processors.

Motivated by the design of an Instruction Systolic Array (ISA) with 1024 processors on one chip [14], an FPU has been designed that meets these requirements. Its bit-serial structure allows for a fine-grained pipelined implementation on a minimal area of silicon. The design has been guided by experiences with a lower-integrated predecessor version of the processor array, which is available as an add-on board for a PC [6].

In the early days of massively parallel machines the individual processors had no integrated FPUs [3]. But when MFlops evolved as one of the most important measures for supercomputer performance, the designers added commercial bit-parallel floating-point processors as coprocessors to small PE-clusters of their massively parallel systems [21,22]. This approach was useful as long as the integrated circuit technology did not allow fitting both, the PEs and the FPUs on the same die. But with the advances in semiconductor technology in the last decade, designers tended to increase the computational capabilities of the individual processors rather than designing parallel systems of floating-point processors.

More recent work has focused on FPU design for FPGAs. In [10] a serial multiplier is combined with a parallel adder in order to achieve high performance with a limited number of CLBs. A similar approach is taken in [9], in order to investigate the resource utilization for the FPU in modern FPGAs. In [8] a technique of parameterizing the FPU-design by the desired precision is discussed. Another adaptive approach has recently been taken in [7]: A generator tool is presented that designs an FPU for FPGA implementation dependent on the input of performance requirements. Even though the performance results are impressive, all these designs are currently not suitable for a massively parallel processor array, since they are not optimized with respect to area and power consumption.

The rest of this paper is organized as follows. Section 2 describes the ISA principle. An overview of the individual ISA processor architecture is given in Section 3. The design methods for implementing efficient floating-point operations on bit-serial hardware are described in

Section 4. Section 5 presents the corresponding execution units. The performance of several floating-point operations is given in Section 6. Section 7 concludes the paper.

2 Principle of the ISA

The ISA [4,5] is a mesh-connected processor grid, where the processors are controlled by three streams of control information: instructions, row selectors, and column selectors (see Figure 1). The instructions are input in the upper left corner of the processor array, and from there they move step by step in horizontal and vertical direction through the array. This guarantees that within each diagonal of the array the same instruction is active during each clock cycle. In clock cycle $k+1$ processor $(i+1,j)$ and $(i,j+1)$ execute the instruction that has been executed by processor (i,j) in clock cycle k . The selectors also move systolically through the array: the row selectors horizontally from left to right; the column selectors vertically from top to bottom. Selectors mask the execution of the instructions within the processors, i.e. an instruction is executed if and only if both selector bits, currently in that processor, are equal to one. Otherwise, a no-operation is executed.

Every processor has read and write access to its own memory. Besides that, it has a designated *communication register (C-register)* that can also be read by the four neighbor processors. Within each clock phase reading access is always performed before writing access. Thus, two adjacent processors can exchange data within a single clock cycle in which both processors overwrite the contents of their own C-register with the contents of the C-register of its neighbor. This convention avoids read/write conflicts and also creates the possibility to perform parallel prefix operations efficiently, e.g. row sum.

3 Processor Architecture

The processor architecture is part of a chip design containing an ISA of size 32×32 [14]. The main components of the processor architecture are a set of 64 data registers, the communication register, a unit for integer addition, logical operations and conditional instructions, a 16-bit multiplier, and a shifter/adder for floating-point arithmetic (see Figure 2). The data word length is 16 bits. Data is processed bit-serially, such that the execution of each instruction takes exactly 16 clock cycles. However, there are macro-instructions formed as series of elementary instructions in order to handle data types consisting of more than one machine word, e.g. double precision floating-point numbers (see Section 4). After receiving an instruction, the processor stores it in the instruction register, decodes the two operand addresses and the destination address, retrieves the operands from the register file, executes the instruction, writes the result back to the destination register, and passes the instruction to the next processor. Since all this is done bit-serially, it can be pipelined on bit level, such that a new instruction can be fetched and processed every 16 clock cycles. For instance, the first bit of the result is computed already after the second bits of the operands have entered the respective unit.

The instructions are 3-address instructions (two operand registers *op0* and *op1*; destination register *dest*) coded in two 32 bits. Several instructions are related to floating-point operations. However, these instructions in general do not make sense on their own, but only within macro-instructions, i.e. a sequence of instructions that realize a certain operation on floating-point numbers. The instruction

`expadd      op0, op1, dest;`

computes the sum of the exponents of two floating-point numbers, as required for a multiplication. Similarly, there exist instructions `expsub` for computing the difference of two exponents and `expsel` for selecting the greater exponent as the result exponent in the case of an

addition of two floating-point numbers. The instructions `shadd` (shift and add), `shsub` (shift and subtract), `nrm` (normalize), `ldmul` (load multiplier), and `mult` (multiply) are used in macro-instructions for computing the mantissa of the result. The instruction

```
mode length;
```

is used for configuration of the floating-point control for handling single precision or double precision numbers. The instructions

```
mtw op0, dest;
expw op0, dest;
```

support the conversion between the IEEE floating-point format [1] and the internal format, forming the mantissa words and the exponent word of the internal format.

4 Floating Point Operations

The processor supports single and double precision floating-point arithmetic. Floating-point registers are implemented by groups of 16-bit data registers and floating-point operations are implemented as instruction macros, i.e. as sequences of elementary instructions.

4.1 Internal Floating Point Format

The special cases for IEEE numbers are coded in all bits of the number representation. For example in order to find out if a number equals infinity the complete mantissa and the complete exponent have to be compared with 0 and -1 , respectively. Therefore, a bit-serial compare to infinity (required for every arithmetic operation) would need at least 32 internal clock cycles for single and 64 for double. To solve this problem a conversion into an internal format is performed. The internal representation of a double X consists of five 16-bit words $X[0], \dots, X[4]$. Here, $X[4]$ contains the exponent of the double, the sign, the kind of representation (absolute value/sign or 2-

complement), and information about possible special cases (infinity, zero, NaN = not a number).

$X[0]$ to $X[3]$ contain the significant (mantissa) of the double, extended to 64 bits.

$X[4]$ Exponent:

Sm	T1	T0	R	Se	Se	e9	e8	e7	e6	e5	e4	e3	e2	e1	e0
----	----	----	---	----	----	----	----	----	----	----	----	----	----	----	----

Sm : sign of the significant, 0: positive, 1:negative

$T1, T0$: encoding of special cases: 00: normal number, 01: zero, 10: INF (infinity), 11: NaN

R : representation of significant: 0: absolute value/sign representation, 1: 2-complement representation

Se : sign of exponent (two identical bits for overflow detection)

$e0, \dots, e9$: binary representation of the exponent E in 2-complement (little endian mode)

64 bit of the significant $M = (X[3], X[2], X[1], X[0])$, where 63 is the most significant bit:

$X[3]$ significant:	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
$X[2]$ significant:	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
$X[1]$ significant:	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
$X[0]$ significant:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

The value of such a double is:

$0 \cdot (-1)^{Sm}$ if $T0=1$ and $T1=0$;

$INF \cdot (-1)^{Sm}$ if $T0=0$ and $T1=1$;

NaN if $T0=1$ and $T1=1$;

$(-1)^{Sm} \cdot 0.M \cdot 2^E$ if $T0=0$ and $T1=0$ and $R=0$;

$(-1)^{Sm} \cdot 0.M' \cdot 2^E$ if $T0=0$ and $T1=0$ and $R=1$, where M' is the 2-complement of M ;

Double numbers can be normalized or not normalized. A positive number and a number in absolute value/sign-representation is normalized iff bits 63, 62, 61 of the significant are 001, i.e. bits 63 and 62 hold the sign and then in bit 61 comes the most significant bit of the significant. As usual in absolute value/sign representation, the significant is positive and the sign

is stored in the S_m -bit of the exponent (see $X[4]$). A negative number in 2-complement is normalized iff bits 63, 62, 61 of the significant are 110. Again, bits 63 and 62 form the sign and it follows the most significant digit of the significant represented in 2-complement.

The four 16-bit words of the IEEE format are converted into 5 words of the internal format. The 52-bit mantissa is stored in four words, where in the first word an explicit 1 is introduced (this 1 is implicit in the IEEE representation). The first two 0's serve as two sign bits of the mantissa. Two sign bits are useful for detecting overflows: an overflow after an addition changes the second sign bit and a right shift for normalization is necessary. The exponent is stored in the last 12 bits of a separate word, also with two sign bits. The value 01 of the sign bits indicates exponent overflow, whereas 10 indicates exponent underflow. In these cases, the number is changed to INF or 0, respectively. The special values zero, INF, and NaN are coded using the tag stored in bit 14 and 13 of the exponent word. In the case of a special value, the actual value of the mantissa is irrelevant. Bit 12 is used internally for indicating if the mantissa is in 2-complement representation after an addition or subtraction. The conversion is supported by special instructions of the processor.

4.2 *Floating Point Addition*

An addition of two floating-point numbers is performed in the usual way: The two exponents are compared, the mantissa of the operand with smaller exponent is shifted to the right appropriately, then the mantissas are added. If any operand is negative and not yet in 2-complement representation, its mantissa is complemented before. The bit-serial organization allows on-the-fly computation with one single half adder. If any operand is 0, its mantissa is set to 0. The result mantissa is in 2-complement representation. The result exponent is the larger exponent of the two operands. The tag of the result is computed depending on the tags of the two operands according to Table 1.

Table 1: Computation of result tags in the case of an addition

	x	0	INF	NaN
x	$x^{1)}$	x	INF	NaN
0	x	0	INF	NaN
INF	INF	INF	INF ²⁾	NaN
NaN	NaN	NaN	NaN	NaN

¹⁾ except the case that the result mantissa is 0, then the result tag is 0

²⁾ except the case that the signs of the two operands are different, then the result tag is NaN

The sign of the result is the sign of the result mantissa, except in the case that any operand is INF, then the sign is equal to the sign of this operand. If the result is 0 or NaN, the sign is irrelevant. Since the operands consist of more than one word (e.g. of four mantissa words in the case of double precision floating-point numbers), shifting of a mantissa by more than 15 bits is realized by displaced addressing of the operands (see Figure 3).

In each case, in order to realize a shift distance of s , in the first step A_{-1} and B_{-1+k} are addressed, where $k = (s \text{ div } 16) + 1$. B_{-1+k} is loaded into the shifter and delayed by $d = 16 \cdot k - s$ bits. In the next step A_0 and B_{0+k} are addressed. While B_{0+k} is loaded to the shifter, the d bits of B_{-1+k} appear at the output of the shifter and are added to A_0 . In the last step A_3 and B_{3+k} are addressed. However, B_{3+k} is an invalid operand, therefore it is cancelled, i.e. set to 0...0 or to 1...1 depending on the sign of the mantissa. If k becomes greater than 4, all B -operands are cancelled, i.e. set to 0...0. In the case that B has the greater exponent, then the difference of the exponents is negative and A has to be shifted. The situation is essentially the same as above with the roles of A and B interchanged.

4.3 Normalization

After an addition, in general the result mantissa is not in normalized form, i.e. there are not exactly two leading 0's or 1's. Several cases are possible: there is one leading digit, two leading

digits (this is the normalized case), three through 63 leading digits, or 64 leading digits. In the latter case, the result is considered to be 0, if the leading digits are 0. In all other cases a left shift by $-1...62$ is necessary so that the mantissa starts with 001 or 110. The exponent has to be corrected by this shift distance. A counter, the *leading digits counter (LDC)*, is used to count the number of leading digits (minus 2) of the result produced during an addition/subtraction or multiplication. The *LDC* also records the first bit of the mantissa, indicating the sign of the mantissa, and detects if the mantissa is 0. Before normalization, the value of the *LDC* is transferred into the shift control register. But now this value has to be interpreted in a different way, since it is the distance for a left shift instead of a right shift as in the case of an addition. However, this can be done using the same hardware for normalization as for addition.

4.4 Floating Point Multiplication

A multiplication of two 64-bit mantissas is performed according to the scheme shown in Figure 4, where e.g. $A_3 A_2 A_1 A_0 B_0$ denotes the 80-bit product of $A_3 A_2 A_1 A_0$ with B_0 . The result would be 128 bit long, however, only the most significant 64 bits can be used for the result mantissa F . Therefore, the above scheme is truncated, in order to reduce the number of 16-bit multiplications. The scheme is illustrated in Figure 5.

Addition of the intermediate results is performed simultaneously with the multiplications in the multiplier, which is actually a multiply-and-add unit. The error introduced by the truncation amounts to at most 3 bits. This seems to be tolerable, because the precision of the internal floating point format is 61 significant bits, whereas the original IEEE format has only 52 bits. If more precise arithmetic is required, the complete multiplications have to be executed. The exponents of the two floating-point numbers are added. The sign of the result is the exclusive-or of the two-operand signs. The tag of the result is computed according to Table 2.

Table 2: Computation of result tags in the case a multiplication

	x	0	INF	NaN
x	$x^{1)}$	x	INF	NaN
0	x	0	NaN	NaN
INF	INF	NaN	INF	NaN
NaN	NaN	NaN	NaN	NaN

¹⁾ except the case of an exponent underflow or overflow, then the result tag is 0 or INF, respectively

5 Floating Point Units

5.1 Multiplier for Positive Operands

As explained in Section 4, bit-serial floating-point multiplication can be implemented by a simple combination of integer addition (for the exponents) and long integer multiplication (for the significants). In this section we describe design of the multiplier for this purpose.

Its original architecture is shown in Figure 6. Before starting the multiplication, one operand x is loaded into a parallel register (the multiplicand register) with the MSB to LSB from left to right. This is controlled by a signal r being high for one clock cycle and low for the rest of the instruction execution. Now the second operand y is input into the multiplier LSB-first. In each clock cycle i one bit y_i of y is processed. It is multiplied with x by computing the bitwise *AND*. The result is added to the current sum, producing a new intermediate result. This result is stored in a carry save representation instead of standard binary representation. This means the intermediate result consists of two binary words: an s -word and a c -word (s for sum and c for carry). The standard binary intermediate result would be $2 \cdot c + s$. This carry save representation avoids the propagation of the carry bits for more than one position.

The next bit y_{i+1} of y is processed in the following clock cycle. Since the result to be computed in this step has to be weighted by a factor of 2 with respect to the old intermediate result we have to shift the old result by one position to the right before we add the new product. Propagating the sum bits to the right neighbour and recycling the carry bits to the same full adder

that has just produced this carry performs this shift operation. The shift operation outputs one bit at the rightmost full adder. This is the i^{th} bit of the final product counted from the LSB to the MSB.

This multiplier design has been used in the parallel computer Systola 1024 [5]. One interesting property of this architecture is its capability to perform *multiply-add* operations. Each bit operation consists of the addition of a new bit product to the intermediate result and a shift operation. Therefore, the interconnection between adjacent full adders form a shift register (the sum register). Now we can use this shift register to add an additional operand. The only thing we have to do is to shift the third operand into the sum register of the multiplier with the LSB first. The first bits of the summand can be shifted in concurrently with the x operand. During the multiplication, in every clock cycle we input one additional bit of the summand while one bit of y is processed.

The multiplier is capable of executing two different instructions, corresponding to the two phases for the multiplication. In the first phase the operand x is loaded into the multiplicand register. We call this instruction LDMUL (load multiplicand). During the execution of a LDMUL-instruction the y -input is forced to be 0. If a multiply-add operation is executed the least significant word of the operand s is shifted into the sum register during the LDMUL-instruction, too. The resulting bits produced by the first LDMUL-instruction are irrelevant.

The second phase is the multiplication with y . The instruction is called MULT (multiply). y is consumed bit by bit as described above. In a MULT-instruction, the x -input is not interpreted. The s -input is used to read the most significant word of the operand s . The output produces the least significant word of the result. After this MULT-instruction the most significant word of the result is still stored in the sum- and carry-registers of the multiplier. In order to output this rest of the result we need to execute another LDMUL-instruction. Therefore, the computation of the last example can be coded in three instructions:

```

LDMUL    x,    slow, don't care;
MULT     y,    shigh, RESlow;
LDMUL    0,    0,    REShigh;

```

Numerous applications in cryptography deal with long operands, e.g. current RSA standards operate with lengths between 1024 and 4096 bits [12]. Such bit lengths can not be processed in one operation of a standard processor. One multiplication with large operands can be decomposed into a sequence of smaller multiplications and additions.

Obviously, an $N \times N$ -bit (N even) multiplication $x \cdot y$ can be constructed of four $N/2 \times N/2$ -bit multiplications as follows:

$$x \cdot y = x_{\text{high}} \cdot y_{\text{high}} \cdot 2^N + x_{\text{high}} \cdot y_{\text{low}} \cdot 2^{N/2} + x_{\text{low}} \cdot y_{\text{high}} \cdot 2^{N/2} + x_{\text{low}} \cdot y_{\text{low}}$$

where $x = x_{\text{high}} \cdot 2^{N/2} + x_{\text{low}}$ and $y = y_{\text{high}} \cdot 2^{N/2} + y_{\text{low}}$

In this case we need four multiplications of half the size and three additions to get one result of full size. If we recursively use this technique we can perform an $N \times N$ -bit multiplication by using an n -bit multiplier. It requires $N/n \cdot N/n$ small multiplications and $N^2/n^2 - 1$ additions.

The bit serial structure of the above multiplier allows for a simple and efficient execution of long multiplications: Only the operand x is chopped into pieces $X_{N/n-1}, \dots, X_0$ where each piece has the length n of the multiplicand. After loading one of these pieces of x using a LDMUL-instruction the complete factor y is processed by a sequence of MULT-instructions. This takes $N/n + 1$ instruction cycles. The result has the length $N + n$. For the complete long multiplication we need to add one such partial product for each piece of x .

To speed up this procedure, we can take advantage of the multiply-add capability of our multiplier. We start with the least significant word X_0 of x . It is multiplied by the complete operand y leading to a result words $RES_{N/n} RES_{N/n-1} \dots RES_0$. RES_0 is the least significant word of the final result. Afterwards we proceed with X_1 . Using multiply-add we compute $X_1 \cdot y +$

$RES_{N/n}RES_{N/n-1}\dots RES_1$. Again we come up with an intermediate result with $N+n$ bits, where the least significant word is part of the final result. In the same way we continue until we come to the most significant word of x . The result of its multiplication with y delivers the rest of the final result. The following example shows the multiplication of two 2-word operands x and y together with the addition with a third 2-word operand s according to the scheme described above.

```

LDMUL      xlow,  slow,  0;    // first stage: load xlow

MULT       ylow,  shigh, RES0; // compute lower result

MULT       yhigh, 0      res1; // compute 2nd partial result

LDMUL      xhigh, res1, res2; // first stage: compute 3rd partial result; second
                                // stage: load xhigh

MULT       ylow,  res2, RES1; // compute 2nd final result

MULT       yhigh, 0,      RES2; // compute 3rd final result

LDMUL      0,      0,      RES3; // compute upper final result

```

Note that there are slots left to add a 4-word operand, however, an overflow might occur. The programmer would have to take appropriate precautions.

This example can be extended to longer operands as described above. In general, the multiplication of two m -word operands requires $m \cdot (m+1) + 1$ instruction cycles.

5.2 Extended Multiplier for Negative Operands

The multiplier of the previous section is capable of multiplying two unsigned integers. With only some minor modifications in the instruction set and in the hardware it is possible to extend it to a signed multiplier where one operand may be negative in 2-complement representation. We make use of a well-known property of 2-complement numbers [2].

Proposition: Let x and y be n -bit numbers in 2-complement representation, i.e. $x, y \in [-2^{n-1}, 2^{n-1} - 1]$ and let x' and y' be the values of x and y interpreted as unsigned integers, i.e. $x' = x$ if $x \geq 0$ and $x' = 2^n + x$ if $x < 0$ respectively $y = y'$ if $y \geq 0$ and $y' = 2^n + y$ if $y < 0$. Let $z = x \cdot y$ and $z' = x' \cdot y'$. Then it holds $z = z' - \text{sign}(y) \cdot 2^n \cdot x' - \text{sign}(x) \cdot 2^n \cdot y' + \text{sign}(y) \cdot \text{sign}(x) \cdot 2^{2n}$, where $\text{sign}(x)$ and $\text{sign}(y)$ denotes the sign-bit of x and y respectively, i.e. $\text{sign}(x) = 0$ if $x \geq 0$ and $\text{sign}(x) = 1$ if $x < 0$.

Proof: We consider the four cases determined by the signs of x and y .

- (i) $x \geq 0, y \geq 0: z = x \cdot y = x' \cdot y' = z'$.
- (ii) $x \geq 0, y < 0: z = x' \cdot (y' - 2^n) = x' \cdot y' - 2^n \cdot x' = z' - 2^n \cdot x'$.
- (iii) $x < 0, y \geq 0$: This case is analogue to (ii).
- (iv) $x < 0, y < 0: z = (x' - 2^n) \cdot (y' - 2^n) = x' \cdot y' - 2^n \cdot x' - 2^n \cdot y' + 2^{2n} = z' - 2^n \cdot x' - 2^n \cdot y' + 2^{2n}$.

In case (iv) the value 2^{2n} can be omitted, because interpreted as a $2n$ -bit number it is equal to 0.

In the above proposition z' corresponds to the result of the multiplication of x and y with the unsigned multiplier of Section 2. The terms $-\text{sign}(y) \cdot 2^n \cdot x'$ and $-\text{sign}(x) \cdot 2^n \cdot y'$ denote the correction steps associated with negative operands to compute z from z' . In order to extend the original multiplier to deal with 2-complement numbers these correction terms have to be integrated into the new multiplier architecture.

Let us assume that x is negative. According to the proposition $2^n \cdot y'$ has to be subtracted from the result z' . This is equivalent to adding the 2-complement of y to the most significant word of z' (which has the weight 2^n). In the bit-serial case, the 2-complement can be computed on-line with the multiplication by using a multiplexor and a half adder. For the implementation of the multiplication with negative x we introduce the instruction **MULTSX**. **MULTSX** is basically identical to **MULT**. Additionally, it sets *xneg* to the sign (stored in the MSB) of x . It is used to select either 0 or $-y$ as additional summand to s . The addition is performed on-line with the input of y and bit-serially delivers $s-y$ at s_{in} . The resulting architecture is shown in Figure 7.

5.3 Floating Point Adder

The floating point adder consists of a shifter, an adder, two units for displaced addressing of the operands, two complement units, and a logic for cancellation of operands. The shifter is a configurable shift register of variable length between 0 and 15 bits (see Figure 8).

For a floating-point addition, the result of the exponent comparison is stored in a special register, the shift control register *SCR*. The highest bit *SCR*[11] indicates if the exponent difference is negative and determines the shift operand. Bits *SCR*[0..3] determine the delay of the shifter. If this is a non-negative number *s*, it is complemented by the XOR-gates and the last 1-bit delay is activated, in order to achieve the required delay of 16−*s*. Bits *SCR*[4..5] are used for displaced addressing of the shift operand, as explained in Section 4. If any of the bits *SCR*[6..10] is different from *SCR*[11], the shift distance would be greater than 64, so the shift operand may be cancelled. A bit-serial adder adds the shifted operand and the other operand. If any operand is negative and not yet in 2-complement representation, it is complemented before the addition.

6 Performance

For single precision floating-point numbers, an addition takes 14 instruction cycles and a multiplication takes 7 instruction cycles. The multiplication consists of the addition of the exponents and the multiplication and rounding of the significands. The internal format allows to detect all exceptions (0, infinity, or NaN) by interpretation of the tag-bits during the addition of the exponents in the first instruction. A truncated multiplication of 32-bit operands requires six multiplication operations: the first loads the least significant half of the first operand into the multiplier. The second multiplies it with the most significant half of the second operand. The third loads the most significant half of the first operand and generates the most significant 16 bits of the intermediate result. In the remaining three instructions, the intermediate result is added to

the product of the most significant word of the first operand and the whole second operand. Table 3 shows detailed instructions for the multiplication.

Table 3: Multiplication of two single precision numbers with the presented FPU. The two operands are stored in registers A[0], A[1], A[2] and B[0], B[1], B[2]. The result is written into the registers P[0], P[1], P[2].

Instructions				Comments
expadd	B[2],	A[2],	P[2];	{ add exponents }
ldmul	0,	A[0],	0;	{ load A[0] }
mulzu	B[1],	0,	X[0];	
ldmul	X[0],	A[1],	X[1];	{ load A[1] }
mulzu	B[0],	X[1],	0;	
mulz	B[1],	0,	P[0];	
ldmul	0,	0,	P[1];	

The addition is more complicated, since the difference of the exponents provides the shift distance of one operand. It is computed in the first instruction again together with the tags for possible exceptions. Controlled by the shift control register (SCR) the indirect addressing fetches the right fractions of the significands to be added. The leading digit counter (LDC) counts the number of leading 0s in order to detect the final shifting distance for normalization after the addition. Finally, the result is shifted left by this distance, and the exponent of the result is decreased accordingly. Table 4 shows details of the addition.

Table 4: Addition of two single precision numbers with the proposed FPU. The two operands are stored in registers A[0], A[1], A[2] and B[0], B[1], B[2]. The result is written into the registers B[0], B[1], B[2].

	Instructions			Comments
expsub	A[2],	B[2],	T[2];	{ compare exponents }
setnf	T[2],	SCR;		{ store difference of exponents in SCR register }
noop				{ wait until SCR is available }
shaddu	0,	0,	0;	{ initialize shifter with 0's and start adder }
shaddu	A[3],	B[3],	0;	{ one address will be modified, the appropriate operand will be shifted and added }
shaddz	A[0],	B[0],	T[0];	{ start LDC and add }
shadd	A[1],	B[1],	T[1];	{ add }
expse10	A[2],	B[2],	T[2];	{ select the greater exponent }
expsub	A[2],	LDC,	0;	{ wait for LDC and adapt exponent: }
expsub	A[2],	LDC,	B[2];	{ create exponent of result }
or	0,	LDC,	SCR;	{ initialize SCR }
nrmu	A[3],	0;		{ start normalization }
nrm	A[0],	B[0];		{ normalize }
nrm	A[1],	B[1];		{ normalize }

The performance of one processor is

$$2 \text{ operations} / 21 \text{ instruction cycles} \times 1 \text{ instruction cycle} / 16 \text{ clock cycles} \times f \text{ clock cycles} / \text{second} = f/168 \text{ operations} / \text{second}$$

With $f = 200$ MHz we have a performance of one processor of 200/168 million floating-point operations/second or 1.2 MFlops; this amounts to 1.2 GFlops for one chip with 1024 processors or to 20 GFlops for a board with 16 chips. For double precision floating-point numbers, an addition takes 18 instruction cycles, a multiplication takes 16 instruction cycles. The performance of one processor is

$$2 \text{ operations} / 34 \text{ instruction cycles} \times 1 \text{ instruction cycle} / 16 \text{ clock cycles} \times f \text{ clock cycles} / \text{second} = f/272 \text{ operations} / \text{second}$$

With $f = 200$ MHz we have a performance of one processor of 200/272 million floating-point operations/second or 0.735 MFlops; this amounts to 735 MFlops for one chip with 1024 processors or to 12 GFlops for a board with 16 chips.

Also more complex floating-point operations can be computed with the presented FPU, e.g. division, square root. Table 5 shows the instruction count for several floating-point operations with the presented FPU.

Table 5: Instruction counts for several floating-point operations in single/double precision.

Operation	Single	Double
Format conversion	6	8
Addition/Subtraction	14	18
Multiplication	7	16
Division	52	101
Square root	92	138

Since instruction count for individual floating operations does not consider format conversion time and communication time within the processor array to fetch the operands, it is often a weak performance measure that does not reflect the behavior of a complete application. Therefore, we will use the instruction count for two typical floating-point applications on the ISA in our performance evaluation (see Table 6). The two applications are 2-dimensional matrix multiplication and 1-dimensional Fast Fourier Transform (FFT).

Matrix Multiplication: The problem is to compute the product of two $n \times n$ matrices $F = A \cdot B$, defined as $F_{ij} = \sum_{k=1, \dots, n} A_{ik} \cdot B_{kj}$. We first assume that $n = 32$ (thus, fitting into the processor array). Suppose that matrices A and B have been loaded into the array. Then, the first column of A , which is in the first column of the array, is broadcast along the rows and stored intermediately

in some single or double register H . Similarly, the first row of B is broadcast along the columns. Now each processor (i,j) has elements A_{i1} and B_{1j} available and can compute $F_{ij} := F_{ij} + A_{i1} \cdot B_{1j}$. The next step is to ringshift A by one column to the left in order to get the second column of A , elements A_{i2} , into the first column of the array, from where they can be broadcast along the rows. Similarly, B is ringshifted along the columns. After n such steps, registers F contain the product of A and B . Multiplication of larger matrices can be easily partitioned using 32×32 matrix multiplication as building block.

Each iteration step consists of two floating-point operations per processor (addition and multiplication) and the ringshift/broadcast communication. The two floating-point operations need 21 instruction for single precision and 34 instructions for double precision. Ringshift/broadcast requires 3 instructions per register on the ISA. Communication costs can be halved by performing a 64×64 matrix multiplication and assigning 2×2 values to each processor. This leads to 30 instructions per iteration step for single precision and 49 instruction per iteration step for double precision, which corresponds to a performance of 0.853 GFlops (single) and 0.522 GFlops (double) receptively. The time required for format conversion can be neglected, since it is only done at the beginning and at the end of the procedure.

FFT: Figure 9 shows the signal flow graph of the FFT of a sequence of n points, here for $n = 8$. It consists of $\log(n)$ stages. In each stage, a complex multiplication with some primitive root of unity, a complex addition, and a complex subtraction is performed at each of the n points. The output of stage k is permuted by interchanging the two halves of subsequences of length 2^k before being input to the next stage. Initially, the sequence of the n points is permuted according to the so-called bit reversal permutation.

The implementation of the 1D-FFT of size $n = 2048$ on a 32×32 ISA assigns two values to each processor. The computation in each stage consists of four (real) multiplications and six (real) additions in each processor. The implementation of the permutation between every two

stages involves routing of data, which is performed either within the rows of the ISA (in the case of interchanging the two halves of subsequences of length ≤ 64) or by interchanging complete rows of the ISA (in the case of interchanging the two halves of longer subsequences). The interchanging of the C -register contents of two halves of m processors is implemented by $m/2$ ringshift operations.

The ten floating-point operations per stage require 112 instructions for single precision and 172 instructions for double precision. The necessary communication per iteration step can be performed on the average within 92 instruction (single) and 153 instruction (double), respectively. This leads to a performance of 0.613 GFlops (single) and 0.427 GFlops (double) for the processor array of size 32×32 .

Table 6: Floating-point performance of the 2-dimensional matrix multiplication and 1-dimensional FFT on the ISA processor array of size 32×32 . Efficiency indicates the achieved performance with respect to the theoretical peak performance, i.e. 1.2 GFlops (single) and 0.735 GFlops (double).

Application	Single (Efficiency)	Double (Efficiency)
Matrix Multiplication (64×64)	0.853 GFlops (71%)	0.522 GFlops (71%)
FFT (2048-points)	0.613 GFlops (51%)	0.427 GFlops (58%)

7 Conclusions and Future Work

In this paper the design of a FPU has been presented. Its bit-serial nature allows for an area efficient and energy efficient implementation. Because bit-serial operations with the IEEE floating-point number representation are difficult to implement, we use an internal floating-point format that allows fast LSB-first arithmetic. Our FPU is optimally suited for a fine-grained

pipelined architecture. That is why it has been used in the processing elements of the KPROC [6], a parallel system with 1024 processors on a single chip.

Most architectures require fast floating-point computation for any price. In case of a massively parallel system on a single chip, there are other criteria that might be even more important than latency: area and power consumption. In this case it turns out that the bit serial approach provides significant advantages in comparison to the bit parallel one. Adder and multiplier can be implemented in a more economic way, exceptions can be detected earlier, and rounding and normalization are relatively simple in comparison to the parallel case. The disadvantage of higher number of clock cycles for one operation is compensated by an increased number of processing elements that fit on one chip.

Due to the advances in FPGA technology it will soon be possible to integrate an array of several hundred simple processors on one single FPGA. An architecture like that could be immediately equipped with floating-point capabilities using the FPU presented in this paper.

References

- [1] *IEEE standard for binary floating point arithmetic*, ANSI IEEE754-1985, New York, (1985).
- [2] J.L. Hennessy, D.A. Patterson: *Computer Architecture: A Quantitative Approach*, Morgan Kaufman Publishers (1996).
- [3] D. Hillis, *The Connection Machine*, MIT Press, Cambridge, MA (1986).
- [4] M. Kunde, et al.: The Instruction Systolic Array and its Relation to Other Models of Parallel Computers, *Parallel Computing* 7 (1988) 25-39.
- [5] H.W. Lang, The Instruction Systolic Array, a parallel architecture for VLSI, *Integration, the VLSI Journal* 4 (1986) 65-74.
- [6] H.W. Lang, R. Maaß, M. Schimmler, The Instruction Systolic Array - Implementation of a Low-Cost Parallel Architecture as Add-On Board for Personal Computers, *Proceedings of HPCN'94, LNCS 797, Springer* (1994) 487-488.

- [7] J. Liang, R. Tessier, O. Mencer, Floating Point Unit Generation and Evaluation for FPGAs, *Proceedings of FCCM'03*, IEEE Computer Society (2003).
- [8] G. Lienhart, A. Kugel, R. Männer, Using Floating-Point Arithmetic on FPGAs to Accelerate Scientific N-Body Simulation, *Proceedings of FCCM'02*, IEEE Computer Society (2002).
- [9] W.B. Ligon, S. McMillan, G. Monn, K. Schoonover, F. Stivers, K.D. Underwood, A Re-evaluation of the Practicality of Floating-Point Operations on FPGAs, *Proceedings of FCCM'98*, IEEE Computer Society (1998).
- [10] L. Louca, T.A. Cook, W.H. Johnson, Implementation of IEEE Single Precision Floating Point Addition and Multiplication on FPGAs, *Proceedings of FCCM'96*, IEEE Computer Society (1996) 107-116.
- [11] M. Meißner, S. Grimm, W. Straßer, J. Packer, D. Latimer, Parallel volume rendering on a single-chip SIMD architecture, *Proceedings of the IEEE Symposium on Parallel and Large Data Visualization and Graphics*, IEEE Computer Society, (2001) 107-113.
- [12] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, *Handbook of Applied Cryptography*, CRC Press (1997).
- [13] B. Schmidt, H. Schröder, M. Schimmler, Tomographic Image Reconstruction on the Instruction Systolic Array. *Computing and Informatics* 20 (2001) 27-42.
- [14] B. Schmidt, M. Schimmler, KPROC – An Instruction Systolic Architecture for Parallel Prefix Applications, *Parallel and Distributed Computing Practices*, 3 (2) (2000) 163-178.
- [15] B. Schmidt, M. Schimmler, H. Schröder, Long Operand Arithmetic on Instruction Systolic Computer Architectures and Its Application to RSA Cryptography, *Proceedings of Euro-Par'98*, LNCS 1470, Springer (1998) 916-922.
- [16] B. Schmidt, M. Schimmler, A Parallel Accelerator Architecture for Multimedia Video Compression, *Proceedings of Euro-Par'99*, LNCS 1685, Springer (1999) 950-960.
- [17] B. Schmidt, H. Schröder, M. Schimmler, Massively Parallel Solutions for Molecular Sequence Analysis, *Proceedings of IPDPS'02*, IEEE Computer Society (2002).
- [18] L.C. Sim, H. Schröder, MIMD-SIMD Hybrid System: Towards a new low-cost Parallel System, *Parallel Computing* 29, (1) (2003) 21-36.

- [19] L.C. Sim, H. Schröder, L.C. Chian, G. Leedham, Fast Solution of large $N \times N$ Matrix Equations in an MIMD-SIMD Hybrid System, *Parallel Computing* 29, (11-12) (2003) 1669-1684.
- [20] M. Taveniku, A. Ahlander, M. Jonsson, B. Svensson, The VEGA Moderately Parallel MIMD, Moderately Parallel SIMD, Architectures for High Performance Array Signal Processing. *Proceedings of IPPS/SPDP'98*, IEEE Computer Society (1998) 226-232.
- [21] T. Thiel: The Design of the Connection Machine, *Design Issues*, Vol. 10, No. 1 (1994)
- [22] F. Valafar, O.K. Ersoy: A Parallel Implementation of Backpropagation Neural Network on MassPar MP-1, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'95)*, Athens, GA, November (1995) 158-166

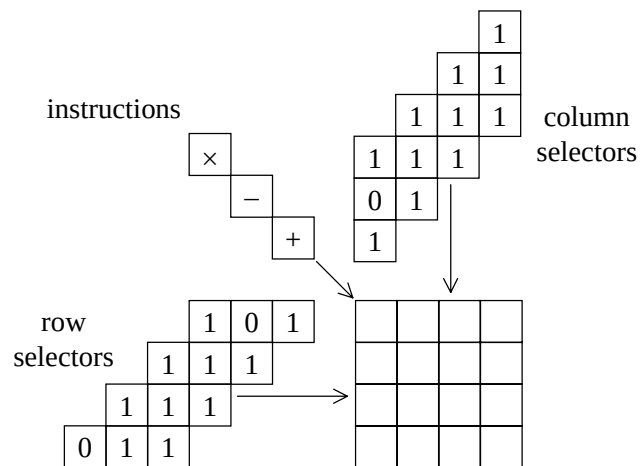


Figure 1: Control flow in an ISA.

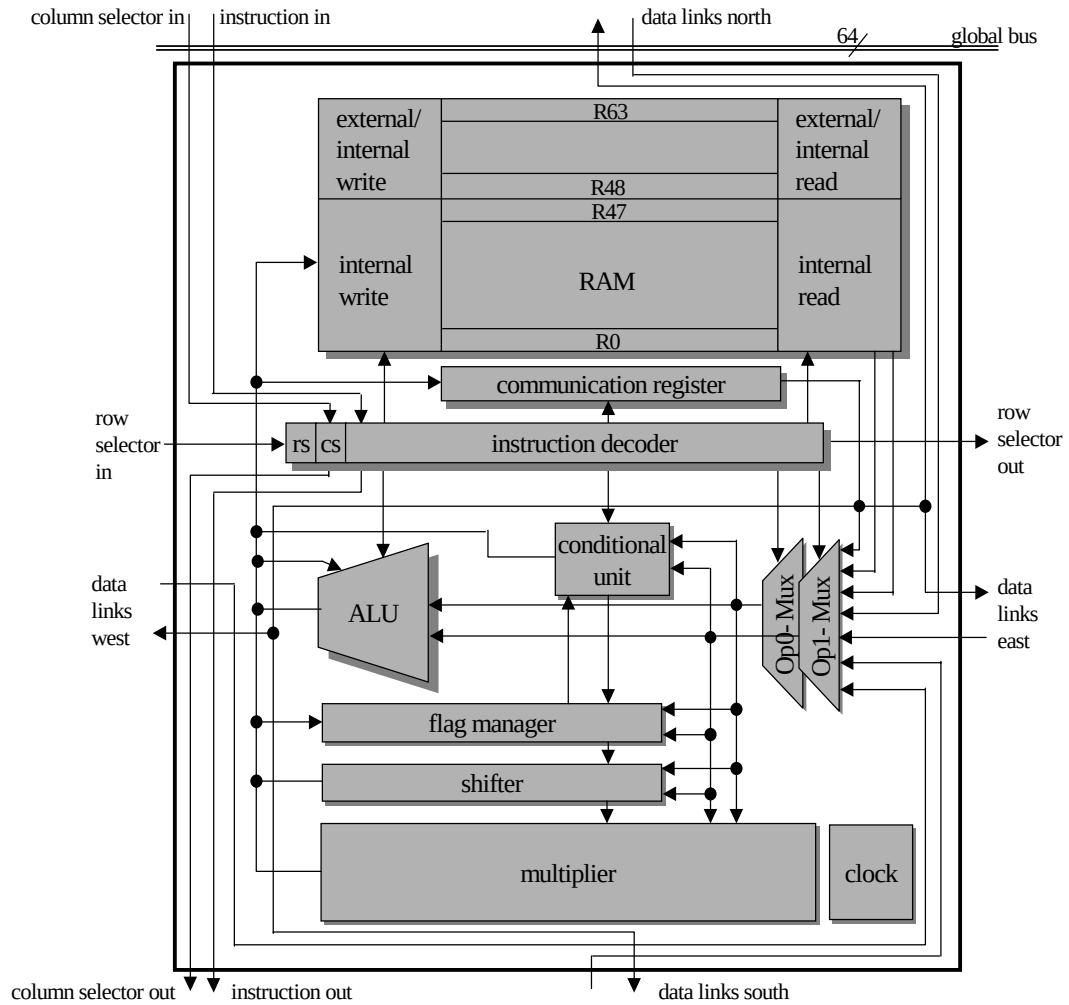
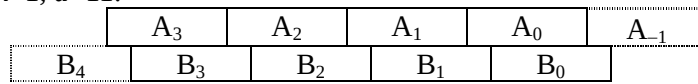
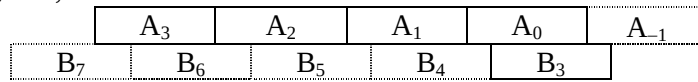


Figure 2: Block diagram of the processor architecture.

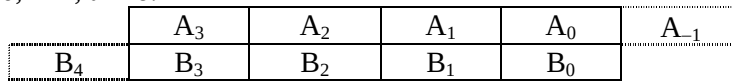
$s=5, k=1, d=11$:



$s=53, k=4, d=11$:



$s=0, k=1, d=16$:



$s \geq 64, k \geq 5$:

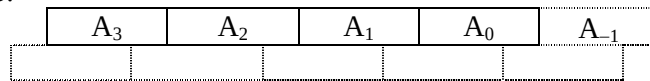


Figure 3: Addition of two 64-bit mantissas.

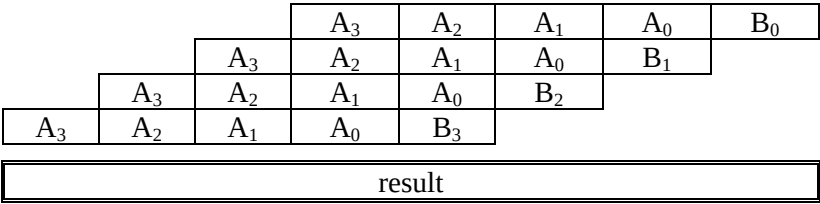


Figure 4: Multiplication scheme for 4-word operands.

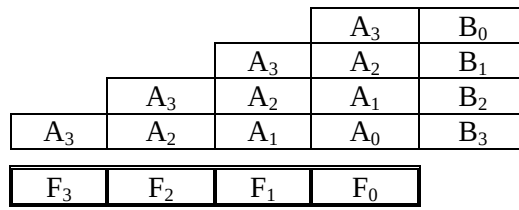


Figure 5: Truncated multiplication scheme for 4-word operands.

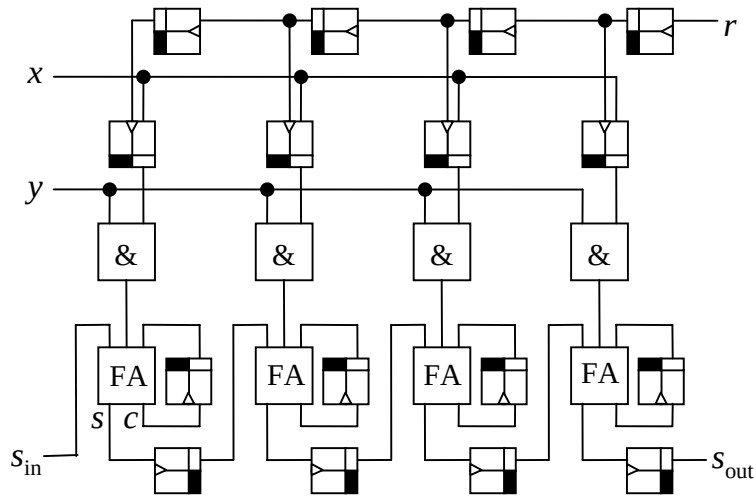


Figure 6: The original multiplier architecture for unsigned integers.

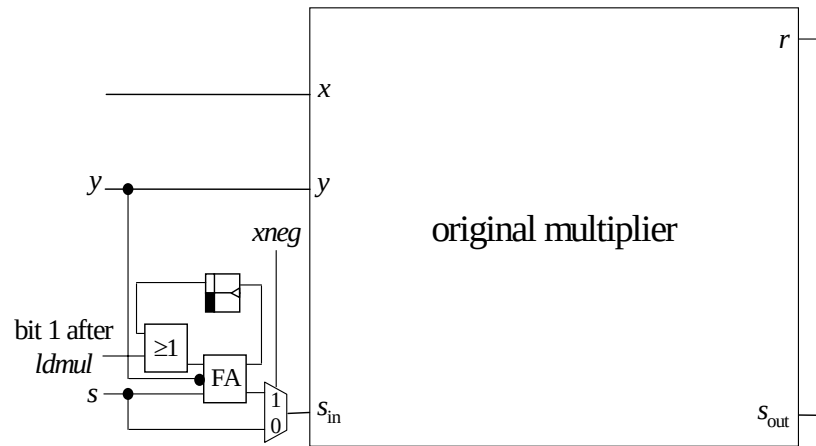


Figure 7: Multiplier for nonnegative y and arbitrary x .

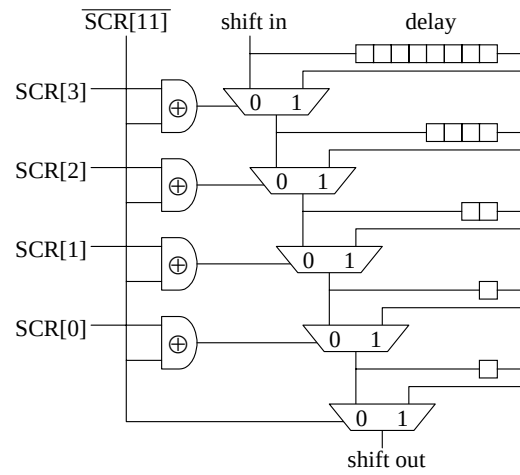


Figure 8: 16-bit shifter.

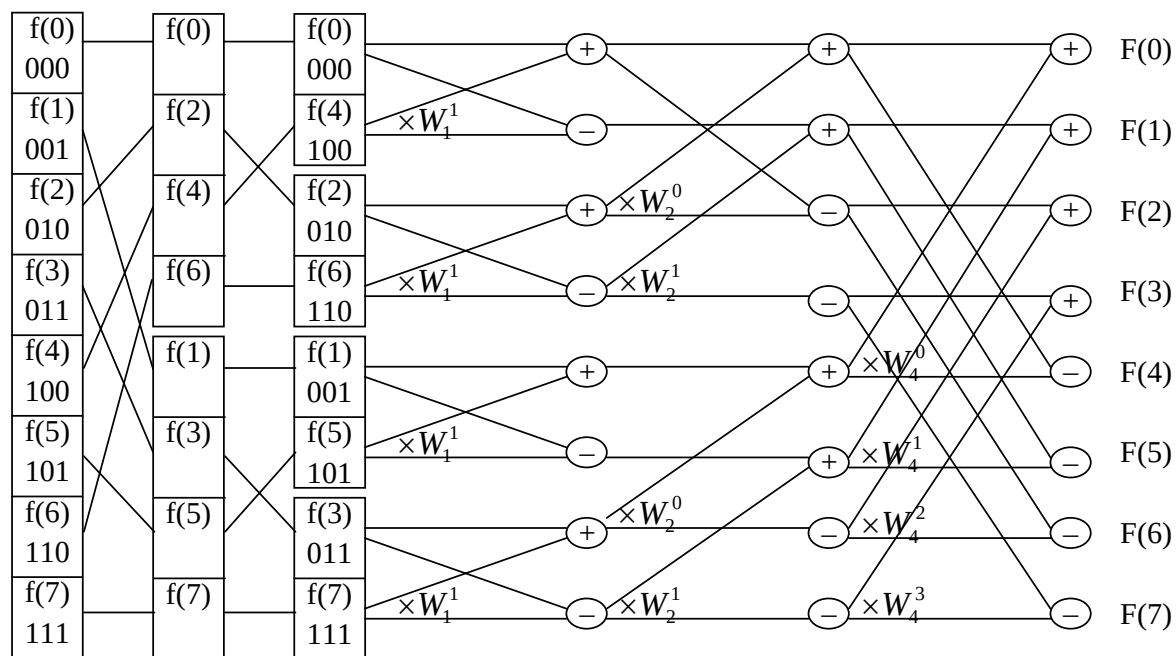


Figure 9: Signal flow graph of an 8-point FFT.

List of Figure Captions

Figure 1: Control flow in an ISA.

Figure 2: Block diagram of the processor architecture.

Figure 3: Addition of two 64-bit mantissas.

Figure 4: Multiplication scheme for 4-word operands.

Figure 5: Truncated multiplication scheme for 4-word operands.

Figure 6: The original multiplier architecture for unsigned integers.

Figure 7: Multiplier for nonnegative y and arbitrary x .

Figure 8: 16-bit shifter.

Figure 9: Signal flow graph of an 8-point FFT.