THE UCSC KESTREL HIGH PERFORMANCE SIMD PROCESSOR: PRESENT AND FUTURE

Francisco Mesa-Martinez, Eric Perlman and Richard Hughey

Abstract— *The UCSC Kestrel parallel processor is a single-board linear array processor with 512 8-bit processing elements. In the process of building the machine, we have touched nearly all aspects of computer engineering, from VLSI layout to board design and debugging, and from device drivers to new algorithm development. The programmable array is primarily designed for several core algorithms from computational biology, on which Kestrel can outperform a workstation by a factor of 20. We have also considered a variety of other algorithms, including graph coloring, computational chemistry, and neural network evaluation.*

Index Terms— *SIMD, parallel processing, computer architecture, undergraduate research, computational biology*

Introduction

The UCSC Kestrel parallel processor is a single-board linear array processor with 512 8-bit processing elements. In the process of building the machine, we have touched nearly all aspects of computer engineering, from VLSI layout to board design and debugging, and from device drivers to new algorithm development. The programmable array is primarily designed for several core algorithms from computational biology, on which Kestrel can outperform a workstation by a factor of 20. We have also considered a variety of other algorithms, including graph coloring, computational chemistry, and neural network evaluation.

Kestrel has had major success both in building a working system and in involving undergraduates in the development of the hardware and software. Several core Kestrel people began working on the machine as undergraduates and later joined our Master's and Doctoral programs. With the constant flow of new undergraduates into the project, we have developed a core set of exercises for gaining experience with this single-instruction, multiple-data stream (SIMD) machine. New students complete several basic programs, such as sorting and polynomial evaluation, and one or two more complicated algorithms, such as dynamic programming sequence comparison. After this introduction to the Kestrel architecture and assembly language, the new researchers then move on to their individual projects.

Department of Computer Engineering, Jack Baskin School of Engineering, University of California, Santa Cruz, CA 95064, {javi,ericp,rph}@cse.ucsc.edu, http://www.cse.ucsc.edu/research/kestrel

Kestrel Academic Impact

UC Santa Cruz describes itself as a research university with an uncommon commitment to undergraduate education. One of the most important aspects of this commitment is the ready incorporation of undergraduates in our research programs. In computer systems development, this can be a tricky process, as undergraduates are either not prepared to design a new computer system or just about to graduate. The success of Kestrel stems from having all initial designs developed and evaluated by a core group of graduate students and faculty. Once the basic architecture was complete, we were able to incorporate growing amounts of undergraduate assistance in chip layout and system software development. The second generation machine has been undergoing a similar evolution — faculty, graduate students and experienced Kestrel undergraduates, designing the system architecture in sufficient detail to break apart and distribute tasks among our new summer colleagues. We have strong hopes that the second generation hardware and software will be as successful as the first.

A brief outline with some of the most important milestones in the realization of the first generation Kestrel is presented (Figure 1). The focus of this project has been the development and implementation of alternative architectures and applications for parallel computing. A project of this nature must involve a wide range of disciplines associated with Computer Engineering. This provides an invaluable academic experience to our students, by offering them the kind of practical and theoretical knowledge that can not be obtained through more conventional academic methods.

Students can get involved with the project in several ways. Some students are invited to join the project after displaying exceptional qualities while participating in any of the courses being taught by the Kestrel teaching staff. Other students join our project as part of a class project or thesis topic. In fact, many of the graduate students who take the advanced computer architecture classes, use Kestrel as part of their final projects, enabling them to participate in a current architecture research effort. Students are also welcome to join and get involved with our team given that they display the necessary motivation and skills. During its lifetime over twenty undergraduate students have participated in the Kestrel group. These undergraduates have worked on many areas frequently not part of the standard curriculum. Eight graduate students have also participated. The project has generated two PhD's and several Master's thesis. Over the years Kestrel has employed two postdocs and two faculty members. The number of undergraduate and graduate students involved in the project

0-7803-6669-7/01/$10.00 ©2001 IEEE

NVIDIA Ex1030
NVIDIA Corp. v. Neural AI, LLC
IPR2025-00606

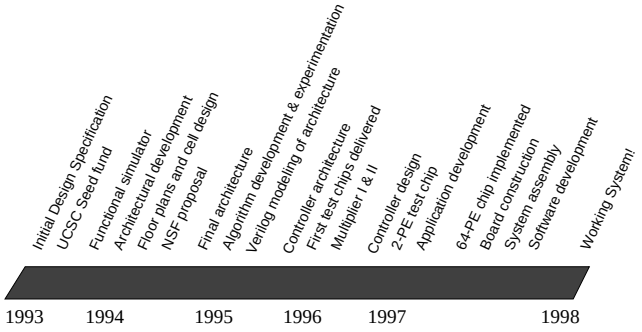


Fig. 1. UCSC Kestrel timeline.

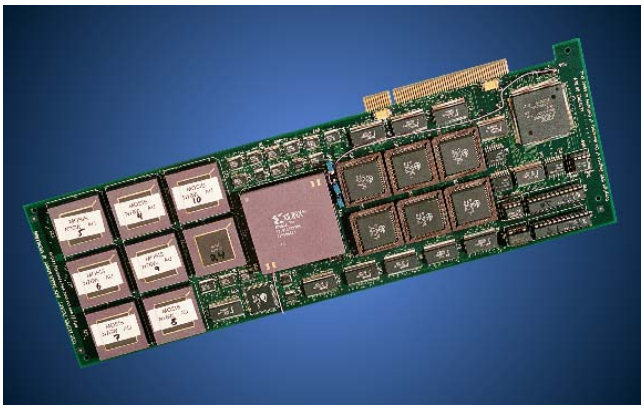


Fig. 2. UCSC Kestrel SIMD computer.

will grow significantly this year as the design of the second generation Kestrel system is going underway.

As a measure of the success of our program, not just in producing a working system, quite a few of our students have received dean's awards due to their research results. Several graduate students have also produced thesis based on work directly related to Kestrel. Due to the large amount of computer design and implementation experience gained by our students, they become extremely employable giving them a clear edge over their fellows. Our students have joined several major engineering firms upon graduation, while others have gone on to further their education by joining graduate programs in engineering.

Kestrel Architecture

The first generation UCSC Kestrel processor is implemented as a systolic array [8] of 512 processing elements (PEs). The system's configuration as a linear array is ideal for such target applications as sequence analysis, at the same time that it provides a simpler and more flexible programming model than more complex multiple-instruction, multiple data stream (MIMD) machines would when applied to the same domain of problems [7]. For the applications that we were envisioning, programmability was more important than large local memories or reconfigurable interconnection fabrics [4]. We believe our SIMD approach to be a more cost effective solution than systems using fully reconfigurable general purpose multiprocessors [5], [6].

Each processing element in the array is 8-bits wide and contains its own local memory and datapath. The Kestrel board also contains several SDRAM banks that implement the instruction memory (IM) as well as the input/output queues (IQm, OQm) [3]. Instruction broadcasting and sequencing is performed by an on board controller, that is also in charge of memory management (Figure 3). The whole system is self contained and implemented as a single full length PCI card (Figure 2).

Like most SIMD processors Kestrel broadcasts each instruction to each PE in the array. Due to the relatively large number of PEs that Kestrel has, and the fact that these PEs

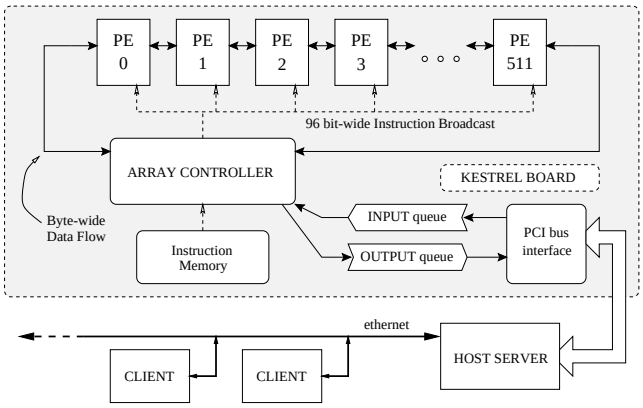


Fig. 3. Kestrel board architecture.

are distributed among several chips, instruction broadcasting can easily become more taxing than the actual time to execute the instruction. In order to solve this bottleneck Kestrel implements a board-level instruction issue and broadcast [4].

In order to handle inter-PE communication each PE shares a file of registers with its left and right neighbors. These register banks are known as Systolic Shared Registers (SSRs) [9]. Although this topology might seem limiting at first, it allows for computation and communication to occur concurrently. When a result is stored in the SSR after the instruction has been completed it is immediately visible to the next PE, thus communication takes place automatically (Figure 4). All addresses to the SSRs are issued globally in order to prevent adjacent PEs from writing to the same bank at the same time. This communication method is both elegant and efficient since it greatly simplifies the design of the PE and allowing most of its area to be dedicated for processing and memory.

In the following sections the architecture of the controller and the processing elements are discussed in further detail.

0-7803-6669-7/01/$10.00 ©2001 IEEE

NVIDIA Ex1030
NVIDIA Corp. v. Neural AI, LLC
IPR2025-00606

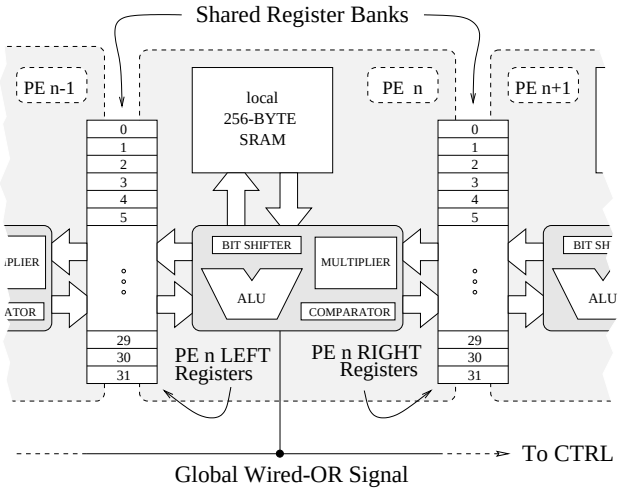


Fig. 4. Kestrel processing element overview.

Controller

The controller is implemented using an FPGA. This device serves as the instruction sequencer for the system. The controller also handles the board's I/O by interacting with the onboard PCI interface chip. The program instructions received from the host computer are stored in a separate instruction memory (IM). The Kestrel controller is programmed by using 54 bits of the 96-bit instruction word. The remaining bits are used as the instruction to be broadcast to the array.

Each cycle, the controller must decide whether or not data is read from the input queue, whether or not data is written to the output queue, and whether or not the immediate field in the instruction should be replaced with data from the controller. A small amount of local memory and logic is used by the controller in order to enable loop counting and the recirculation of data among the PEs. When a critical condition is reached by the system the controller will generate the corresponding interrupt to inform the host about the board's condition.

Processing Element

The heart of the Kestrel architecture is its custom processing element (PE). Each Kestrel PE has the following elements in its datapath: 8-bit ALU and comparator, 8×8 multiplier, bit shifter, and result selector (Figure 4). All the elements within the datapath are designed to complete one operation per cycle. The inclusion of a multiplier and some extra functionality in the ALU was made in order to expand the scope of possible applications for the architecture.

ALU

The ALU takes two 8-bit operands (A,B) and a carry bit as inputs. The A operand always comes from a register while the B operand has several possibilities. The ALU produces an 8-bit result R and a carry output bit, the result can be stored back into a register, and the carry bit can be stored in a latch for future use.

Multiplier

Unlike most of the elements in the datapath the multiplier unit is 16-bits wide, due to the fact that it takes two 8 bit operands (A,B) and produces a 16-bit result R. The result is stored using two 8-bit registers, a normal result register holds the least significant byte, whereas a special register (multiHi) stores the higher order byte [4].

The Kestrel multiplier implements a Modified Booth multiplication algorithm. By converting some of the half adders in its design to full adders, the multiplier is able to complete not just a multiply but also to perform two accumulate functions in a single cycle. The fact that this unit can complete a multiply-accumulate-accumulate each cycle allows our design to offer a significant advantage when dealing with multiprecission multiplications [11].

Bit Shifter

In order to better handle data manipulation and resolution of conditions, a bit shifter is included in the datapath. The bit shifter is implemented as a simple 8-bit shift register that can shift its data left or right by one bit. The conditional processing enhances the ability of each PE to evaluate nested conditions based on local values. If not implemented carefully the functions needed to resolve conditions in SIMD machines can take more cycles than the outcome functions themselves [4].

In the Kestrel architecture, each clause of a conditional is broadcast to the whole array so that the PEs that test positive for the condition can be kept activated and can proceed to execute the outcome code, while those PEs that fail the test are inactivated.

Comparator

This unit compares the result from the ALU with a third operand C. The maximum or minimum element from the evaluation can be selected as the result R for the instruction. There are three supported types of top-down comparison: unsigned comparison, modulo 256 comparison and signed comparison. The comparator allows the Kestrel PE to deal with multiprecission operations in fewer cycles than more conventional architectures.

Result Selector

The last element in the data path, the result selector, decides the final 8 bits that will be stored as the result for the current operation. For operations involving the ALU a selection can be made between the actual result R or the operand C from the instruction. The result can be forced to one or the other of these values, or it can be chosen by a flag in the ALU, comparator, bit shifter, or the shift register from a neighbor PE.

SRAM

Each PE contains 256 bytes of local static random access memory (SRAM) that can be used as local storage. The PE

NVIDIA Ex1030
NVIDIA Corp. v. Neural AI, LLC
IPR2025-00606

can access its SRAM in two modes, absolute and indexed. In absolute addressing the address is the immediate 8-bit value specified in the instruction. For indexed memory accesses the immediate value and operand C are added. Each PE has its own decoder and address generator to access its private SRAM.

The size of the local memory was determined by various factors. First, our PE has 8-bit addressing space that puts a specific limit of 256 words to its addressable memory. Second, the size of the SRAM was determined by the best possible tradeoff between number of PEs per chip and the size of local memory. For most of the target sequence analysis applications 256 bytes of memory per PE seemed to be enough [7].

Second Generation Kestrel

In order to increase the performance of the current system our research group has started the design and implementation of the second generation Kestrel SIMD computer. Due to the large cost associated with the design and fabrication of custom PE chips, the new system will be implemented using the same PE technology. The design effort is being carried out by a new team of one graduate and eight undergraduate students.

The most obvious choice for a quick performance gain was to increase significantly the number of PEs on board, thus the new Kestrel computer will support 1024 PEs versus the 512 that are present in the current implementation. The new revision of the system also has an increased amount of local memory, for both the instruction memory and the I/O queues. The fact that the controller is implemented using a single FPGA allows for continuous revisions of its design without any physical modifications needed to be done to the final production board during its lifespan.

In order to take full advantage of the PE chip's actual speed the second generation Kestrel board will be implemented using a pipelined controller. The target speed for the pipelined controller is 100 Mhz while the execution units will be running at 50Mhz. By using double the number of PEs running at double the speed of the first generation system, the second generation Kestrel will have a 4 fold performance increase over the current implementation.

Kestrel Pipeline

The new controller is implemented using a four stage pipeline. The duties of the new controller are a superset of the ones the first generation controller has to perform. The introduction of pipelining will allow for more efficient and flexible programming models. The four pipeline stages are discussed in more detail in the following subsections.

IF: Instruction Fetch

The controller fetches the instruction from the instruction memory (IM) pointed by the current PC value. Jumps and conditionals must resolved before generating the new PC value. There are two types of jumps: data-dependent jump instructions that have to wait until the completion of the WB stage, and uncoditional, counter, and return jumps that can be resolved directly at this stage.

ID: Instruction Decode

Usually the only reason to stall the pipeline in ID is if a late jump was passed on from the IF stage. If this condition arises the controller will stall the instruction currently present at the IF stage until the current late jump clears the WB stage.

If no other problems are reported, the instruction latched from the previous stage is then decoded and passed to the next stage. Immediate values for the PE array are also generated at this stage. Branches from IF must also be resolved now.

EX: Instruction Execution

During EX the array instruction, that was broadcasted at the end of ID, is ready to be executed by all the active PEs. The PEs access their local data at the beginning of the cycle, calculate and then store the result. At the end of the stage the output from the array is then latched by the controller.

WB: Write Back

The result produced by the array are written either to a temporary memory location (MDout) or to the output queue (OQm) which will allow the data to be shipped back to the host via the PCI bus. The controller performs several memory management operations, such as checking the current state of each queue and register file visible to the controller. If one of these memory locations might be reaching their storage limits, the controller will raise an interrupt.

System Software

An important part of the Kestrel project is the development of a supporting software framework that will provide the end users with both a quality development environment, and a layer of abstraction from the hardware. Most of the development for the kestrel drivers and remote application server for Kestrel are being carried out by two undergraduate students.

Currently all programs for the Kestrel processor must be written in *kasm*, the Kestrel assembly language. The requirement for programs to be written in assembly has limited much of the use of the Kestrel processor to applications developed at UCSC, most notably the Smith & Waterman search [13]. Work on a compiler has started and will make writing Kestre software easier. The compiler will use a language similar to C and it is inspired by previous successful SIMD compilers such as C* [12] and MPL [1], [2]. Efficient SIMD compilers are an active area of research in our group, we are studying efficient partition and scheduling strategies for systolic arrays [10], [8].

Running Code on Kestrel

The software developer has two ways of running programs on Kestrel, either directly on the actual Kestrel hardware or on a

0-7803-6669-7/01/$10.00 ©2001 IEEE

NVIDIA Ex1030
NVIDIA Corp. v. Neural AI, LLC
IPR2025-00606

fully functional, byte compatible, simulator that can be run on a normal workstation.

Kestrel programs are loaded through a local area network to one of several machines that have Kestrel boards installed, these machines act as compute servers for the client workstations. Input data is sent through the same connection, as are the results. The original server was based on an example network application shell for Windows NT. This script was modified to communicate with the board directly over the PCI bus. Due to its inherent lack of scalability, this server is being phased out as the project moves towards using Linux and Unix host systems. This move was prompted in order to decrease costs and enable students to interact with the internals of the operating system in order to implement higher performance drivers and server subsystems.

Server and Driver Architecture

For the implementation of a new Kestrel server, it was decided that Unix would be a much better platform, allowing easy remote access to the server. The driver and server portions are now separated to allow easy porting to other platforms. The driver, which is loaded into the Unix kernel, handles low level communications with the processor board over the PCI bus. The driver deals with issues such as keeping the input queue full and output queue empty. Kestrel drivers are currently available for Linux 2.2 kernels and OSF. The driver is responsible for servicing the board's interrupts, generated by the Kestrel controller, which usually indicate an empty input queue or unflushed output queue (Figure 5).

The server continually listens to the network and allows remote users to load and run Kestrel programs. Unlike the old server, the new one has support for loading multiple programs onto Kestrel at the same time, changing the active program, and sending data to and from the board in small pieces. The client communicates with the server through a TCP connection. The connection is initiated by a function to get a Kestrel board, which communicates with the directory server to find an available board.

A client library was created to allow a C programmer to communicate with the Kestrel board from a remote machine. The interface allows a remote program to communicate with the board while it is running, making it possible to load different programs and feed it different data depending on the results of previous queries. It can also allow processing on both the Kestrel board and remote machine at the same time. This interface works in C++, with a Java version forthcoming.

Remote Application Design

While the Kestrel server allows remote users to connect to Kestrel via a network connection to run programs, it still does not give remote users the same performance that would be expected, if the board was physically located in their workstation. The network can prove to be a major bottleneck for applications that need to either send or receive large amounts

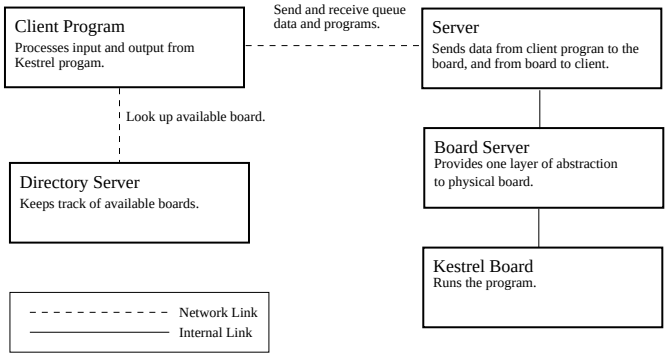


Fig. 5. Kestrel software architecture.

of data. To solve the problem, we are developing an enhanced server that allows programs written in Java to run directly on the Kestrel host machines.

This Work is based based on a prototype created by researchers at UCSF, who wrote a Java based client-server application for doing Smith & Waterman searches directly on a Linux machine with a Kestrel installed. Under this approach each host will have a local library of Java applications, supporting Kestrel code and databases.

The application server is being implemented in Java, using its ability to dynamically load code, and providing a security model allowing strict limits to be placed on the foreign code. The Kestrel application server runs inside a java virtual machine on each server. The server interfaces with the Kestrel board through the Kestrel server using the Java version of the client library. The application server listens to the network, and has its own communication protocol. A remote user must load a program class, either over the network or from the local file system. This program is responsible for loading the appropriate Kestrel programs to the board and parsing the data between it and the remote user.

Having many commonly used programs on the servers will make the Kestrel board much more accessible for those researchers wanting to use the system for any common task suitable for a high performance SIMD machine.

Concluding Remarks

The design and implementation of a working high performance computing system like Kestrel has proven to be a challenging and complex task. The development cycle for this project has involved a multitude of areas in computer engineering, such as core architecture design, system design, VLSI, routing and placement, board level design, FPGA synthesis of large systems, testing, VHDL, device driver and system software design. This sheer breadth in the scope of disciplines has allowed the students involved to obtain a unique learning experience that has provided them with a large set of problem solving skills. The students involved in our project have demonstrated an excellent technical and practical prepa-

NVIDIA Ex1030
NVIDIA Corp. v. Neural AI, LLC
IPR2025-00606

ration for future careers in both academia and industry.

Kestrel has been used for several years in our graduate computer architecture, parallel processing, and computational biology courses. The current design and implementation of the second generation Kestrel system has boosted significantly the amount of students currently involved with the project. Once the new systems are complete, we hope to use Kestrel in a new undergraduate parallel architecture and programming course as a means of including novel computer architectures and system designs in the undergraduate curriculum at UCSC. It is our hope that with these new boards and updated system setting, students at many other universities will have a chance to explore the power and elegance of SIMD computing.

Acknowledgments

Our work in the Kestrel project is supported in part by NSF grants EIA9905322 and DBI9808007.

References

[1] J.D. Becher and K.M. Hansen. Mpl: A data parallel c. *Technical Report MP/O-01.91, MasPar*, January 1991.

[2] T. Blank. The maspar mp-1 architecture. *The 35th IEEE Computer Society International Conference*, pages 20–23, February 1990.

[3] David Dale, Leslie Grate, Eric Rice, and Richard Hughey. The ucsc kestrel general purpose parallel processor. *Proc. Int. Conf. Parallel and Distributed Processing Techniques and Applications*, pages 1243–1249, June 1999.

[4] D. M. Dahle et al. Kestrel: Design of an 8-bit simd parallel processor. *IEEE Proc. 17th Conf. on Advanced Research in VLSI*, 1:145–162, September 1997.

[5] E. Miya et al. Multiprocessor/distributed processing bibliography. *Computer Architecture News (ACM Siggraph)*, 13(1):27–29, 1985.

[6] J. Elder et al. Issues related to mimd shared-memory computers: The nyu ultracomputer approach. *proc. 12th Int'l Symposium on Computer Architecture*, pages 126–135, June 1985.

[7] J. D. Hirschberg, R. Hughey, K. Karplus, and D. Speck. Kestrel: A programmable array for sequence analysis. *IEEE Proc. Int. Conf Application-Specific Systems, Architectures, and Processors*, August 1996.

[8] R. Hughey. Programming systolic arrays. *Proc. Int. Conf. Application-Specific Array Processors, IEEE Computer Society Press*, 1:604–618, August 1992.

[9] R. Hughey and D.P. Lopresti. B-sys: A 470-processor programmable systolic array. *Proc. Int. Conf. Parallel Processing*, 1:580–583, August 1991.

[10] Keyrell and Paris. Activity counter: New optimization for the dynamic scheduling of simd control flow. *Proceedings of the 1993 Int. Conference on Parallel Processing*, 2:184–187, August 1993.

[11] E. Rice and R. Hughey. Multiprecision division on an 8-bit processor. *Proc. 13th IEEE Symposium on Computer Arithmetic*, pages 74–81, July 1997.

[12] J.R. Rose and Jr G. Steele. C*: An extended c language for data parallel programming. *Proceedings of the Second International Conference on Supercomputing*, pages 2–17, May 1987.

[13] T.F. Smith and M.S. Waterman. Identification of common molecular sequences. *J. Mol. Biol.*, 147:195–197, 1981.

NVIDIA Ex1030
NVIDIA Corp. v. Neural AI, LLC
IPR2025-00606
Page 6 of 6

View publication stats