Yet Another Survey on SIMD Instructions

Armando Faz Hernández
armfazh@ic.unicamp.br

June 4, 2013

Abstract

Data level parallelism approach enables a potential speed-up on applications that works with (almost) independent data sets. Recently, parallel processing has been growing and has being commonly used in different applications, such as image/video processing, visualization, scientific computation, etcetera. The latest processor's micro-architectures come equipped with special units that process an operation over a set of data, on a SIMD fashion. In this document, we research main features offered by different instruction sets that perform parallel operations. We focus on various instruction sets designed for either desktop/server computers such as: SSEx, AVX and FMA, and also for mobile and embedded computers with ARM-based processors, which supports the NEON instruction set.

Key words: Data level parallelism, SIMD, vector instructions, NEON, SSE, AVX, ARM.

1 Introduction.

The design of parallel computer architectures has been evolved, since the firsts implementations, in the middle of 1980's, when RISC (Reduced Instruction Set Computer) micro-architectures emerged, they exploited parallelism among independent instructions, the so-called ILP (Instruction Level Parallelism) [18]. RISC-based computers where predominant during more than fifteen years, at that time, many optimizations were developed to extract parallelism between instructions either statically, i.e. at compilation time, and also dynamically by the use of dedicated hardware and sophisticated algorithms.

For the middle of the 1990 decade, processors' manufacturers focus on extracting parallelism over data rather than instructions. This approach considers to perform operations over a set of data instead of a single unit of data. In particular, when an application processes the same operation for every unit on a large set of data, then we have a symmetric computation over data, also known as single instruction multiple data processing (SIMD).

In the SIMD approach, computer's micro-architecture provides additional wider registers, instructions and special processing units that are able to apply one instruction over a set of data. The first use of this approach was in vector supercomputers around 1970 such as Illiac-IV from Illinois University [5], the CDC Star processor [19] and the ASC processor from Texas Instruments [24]. All of them share common characteristics large vector registers storing more than 64 floating point elements each one, multiple functional units and configurable set of registers. One of the most successful implementations used in high performance computing was the Cray supercomputer [20].

Processors' designers started to integrate SIMD features in two flavours, first with the inclusion of internal functional units that are able to execute an instruction over a wider set of registers, and secondly with the addition of multiple core processors on chip. The first desktop user processor with support for SIMD was released in 1996, at that time, not only scientific applications were benefit, but also the increasing image, sound and video processing were present by the emerging multimedia applications. Since then several SIMD instruction sets were developed and nowadays, advanced instructions has a high impact in graphics acceleration, data encryption, advanced image processing, and further applications.

The purpose of this document is to expose different implementations of SIMD instructions present in the latest processors. We focus on traditional desktop/server micro-architectures and also in the low

power consumption processors, in particular, the ARM architecture. Both micro-architectures provide dedicated hardware units and instruction sets that are able to work in a SIMD fashion.

The document is structured as follows: on section 2 we introduce formally the SIMD processing, detailing its characteristics and differences among other parallel approaches. Section 3 describes chronologically the release of SIMD instruction sets for desktop computers and in section 4 for the ARM architecture with its powerful NEON unit. Section 5 discusses about the actual development when the use of SIMD instructions is required. Finally, section 6 gives the final remarks of this survey.

2 Single Instruction Multiple Data Approach

In 1966, Michael Flynn characterized computers through a simple abbreviations used until nowadays [9], he figured out how parallelism is processed over instructions streams and over data streams, thus every computer can be classified into one of the following categories:

- *Single instruction stream, sigle data stream (SISD).*
- *Single instruction stream, multiple data streams (SIMD).*
- *Multiple instruction streams, sigle data stream (MISD).*
- *Multiple instruction streams, multiple data streams (MIMD).*

Most of current computers overlaps some categories due to implement different characteristics in the same architecture. For purpose of this document we will focus on the SIMD category.

SIMD computers take advantage of inherent parallelism over data that are being computed. In that sense, there are three different approaches that allows to implement a SIMD computer:

- **Vector architectures**. As was mentioned above in the introduction, supercomputers was the most successful application of SIMD processing. They present large register files storing over than 64 floating point elements. Normally vector architectures has long memory latency to load or store data between registers and memory, resulting in high bus memory occupancy.
- **SIMD for multimedia**. SIMD for multimedia was specially dedicated to increase the performance of emerging multimedia applications, i.e. when user interfaces evolved from command line consoles to graphical user interfaces, where images are the central elements to process. Furthermore, video and sound were a common factor in contemporaneous applications.
- **Graphic processing units**. Dedicated hardware was required to deal with advanced graphic processing in visualization and game industry. In order to increase the frames-per-second (fps) visualized in graphic applications, floating point units included on-chip were not enough to achieve such speedings, because of that specialized hardware was developed to support fast execution and high parallel processing over image rendering.

These three principal branches of SIMD computers can be found nowadays. However material presented in this document is relevant to SIMD for multimedia, we describe different instruction sets in 64-bit architectures such as Intel and also in low energy processors such as ARM architecture.

3 Intel and AMD instruction sets

In this section, we are going to describe in chronological order the development of SIMD multimedia instruction sets. Initially, Intel was the pioneer in the inclusion of dedicated instructions for SIMD computation, after that AMD released a series of instruction set that converge in advanced set of instructions that can easily used at by end user programmers.

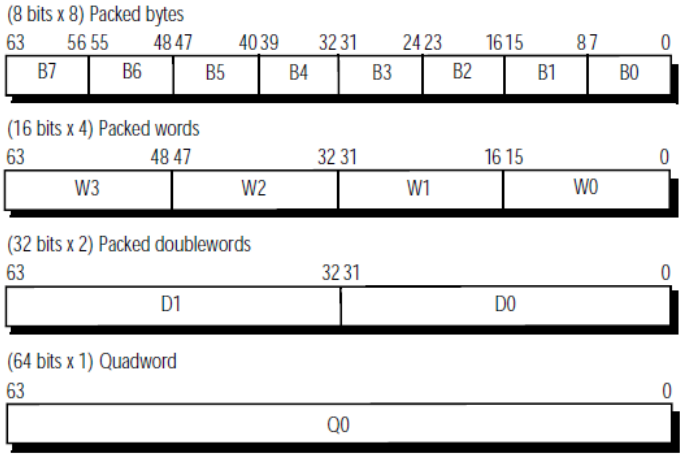

Figure 1: This figure shows the versatility of registers to pack data of different size.

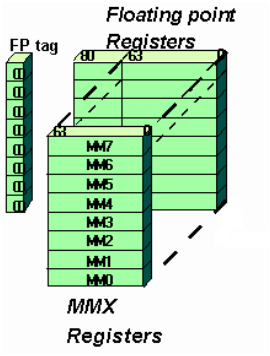

Figure 2: This graph shows how MMX registers are overlapped over floating point registers used by FPU.

3.1 Multimedia extensions MMX

The following two applications motivated the creation of dedicated units that converged in MMX extensions. First, in many graphics systems use 8-bit integers to represent a pixel's colour in RGB mode[1] and a additional 8-bit integer for transparency, pixel processing then requires 8-bit integer arithmetic. Second, audio samples need around 16 bits to be processed by sound and video applications. Intel recognized that these kind of applications perform the same operation over a vector data as is extensively described in [15, 4].

Construct a dedicated functional unit to perform these processing was expensive at that time, however a cheaper idea consists in partitioning the carry chains within a 64-bit ALU, which result in performing simultaneous operations on short data vectors.

Intel introduced in 1997 the MMX instruction set, which stands for *MultiMedia eXtension* [11]. The MMX instruction set adds 57 new instructions and a 64-bit data type (`__m64`). In addition, there are eight 64-bit MMX technology registers, each of which can be directly addressed using the register names `MM0` to `MM7`. The MMX instruction set is based on the concept of packed data types, which means that instead of using the whole register for a single 64-bit integer, two 32-bit integers, four 16-bit integers, or eight 8-bit integers may be processed concurrently, as shown in figure 1. A disadvantage present in MMX technology is that 64-bit registers are overlapped with those of FPU unit, see figure 2, then programmers only can works in one mode (floating point or integer) and switching between modes incurs in performance loses.

In [14], Intel exposes the impact of the inclusion of MMX instruction set that improves the performance in three different aspects: integer, vector and floating point processing.

3.2 3DNow!

In 1998, AMD release 3DNow! technology in order to support common arithmetic operations on single precision floating point data [17]. The first implementation in a AMD-K6 processors contains 21 new instructions in SIMD fashion for floating and integer operations. 3DNow! share the same register file than MMX, carrying the same penalty when moving from FPU registers to MMX/3DNow! registers, it was not possible to be used simultaneously.

3DNow! was not enough popular as the Intel versions of SIMD instructions, because of that AMD in 2010 decided to deprecate 3DNow! instructions for further processor developments [3].

[1] RGB stands for Red-Green-Blue base colors.

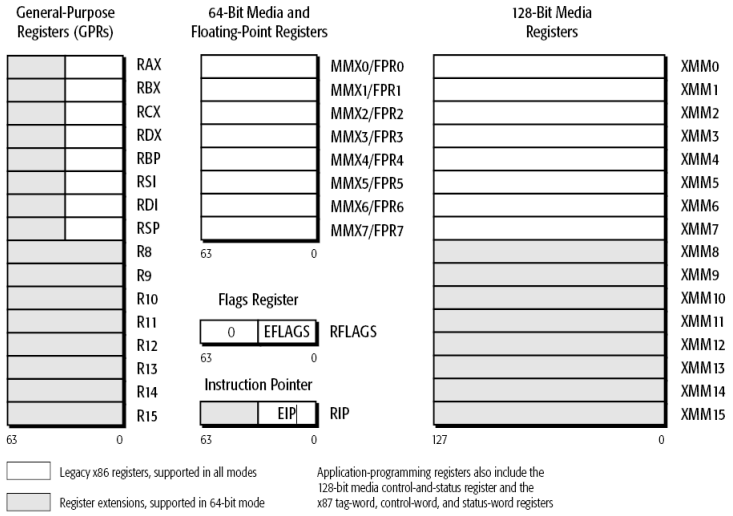

Figure 3: Register set of x64 architecture of an AMD processor.

3.3 The SSE era.

Intel identifies two main problems in MMX technology: first, MMX overlaps the floating point register from FPU unit, which unable the use of both instructions, and secondly MMX only works for integer arithmetic. Due to in 1999, Intel comes with a new release of SIMD instructions called *Streaming SIMD Extensions* (SSE), which contains 70 new instructions mostly for floating point arithmetic.

SSE also introduces a new register set with eight 128-bit registers called `XMM0` to `XMM7`. These registers can be accessed by `__m128` data type and are able to store four single precision floating point values.

In order to deal with high latencies in caches misses, SSE includes 8 new instructions designed to include the ability to stream data to memory without polluting the caches, and to pre-fetch data before it is actually used.

In [1] shows an application that takes advantage of the use of SSE instructions in the matrix multiplication problem, they achieve a speed up of 2 times faster than the state-of-the-art reported in 2001. In [22] Takahashi et. al. report that with the use of SSE instruction set they can achieve 2.25 times faster than non-SSE implementations when solving the parallel LU-decomposition.

In 2000, AMD launched the x64 extension that increase the number of `XMM` registers from 8 to 16, now from `XMM0` to `XMM15`. In figure 3 shows the full set of registers present in the AMD x64 architecture. With the release of Pentium 4 processor (2001), Intel announced the **SSE2** which improves the capabilities of SSE. SSE2 adds 140 new instructions with the main characteristic that SSE2 is able to work on 64-bit floating point elements, computing values with double precision allows to emerging 3D games become more popular. However, Intel was also interested in the scientific field, for example for CAD (*Computer Aided Design*) applications.

Although SSE2 instructions operate in double number of elements than MMX, the performance is roughly the same, because accessing to misaligned data incurs in significant penalties. In order to solve this issue **SSE3** was released in 2004 with instructions that are able to load/store data from unaligned memory addresses. Another characteristic of the inclusion of 13 new instructions in the SSE3 instruction set was the capability to work horizontally in the register, i.e. perform some operations with unit elements loaded within the register, as opposed to previous implementations which process data in vertical way.

For 2006 Intel released the *Supplemental Streaming SIMD Extensions 3* (**SSSE3**), which contains 16 new instructions, two of them perform multiply and add operation, six that evaluate absolute values and two that align data from two operands. In [10] shows how SSSE3 improves the performance when is used to compute the population count in a register, i.e. the number of bits set in a register, this operations is heavly used in the referred application.

Intel **SSE4** consist in a set of 54 instructions, the first 47 known as **SSE4.1** and the last 7 known as **SSE4.2**. Unlike former instruction sets, SSE4 was not directed to the multimedia applications, but fast string processing such as string comparison and native population count. It also contains a specific instruction for error correcting codes, in particular the CRC-32 version [7]. For further details on SSE4 see [13].

The SSE era consist in more than 200 instructions that enables SIMD processing since 1996. The overcoming of processors with integrated SIMD functional units increases performance on image, sound and video applications achieving that end user computers has powerful computing system similar to the scientific computations in the beginning of the SSE instructions.

3.4 The arrival of AVX.

In this section we are going to describe a more advanced technology which involves the use of bigger registers. This is the case of *Advanced Vector Extensions* (**AVX**) released in 2011 on Sandy Bridge architecture. Intel decided to move computations to a wider registers, then 16 256-bit registers were added, called `YMM0` to `YMM15`, the 128 less significant bits are overlapped with the `XMM` register set as is shown in figure 4.

Another feature of AVX is the possibility to write three operand code, in other words, it is possible to specify a destination register beyond the input registers. This feature is also known as non-destructive operands and enables the use of more flexible code for assembler programmers. An important aspect in the use of AVX is the new VEX encoding scheme, which is used to extend the space of operations code in the future architectures and now allows instructions to have up to five operands. This new coding scheme support the legacy SSE instructions just by adding the VEX prefix in the instruction. For efficiency reasons it is recommended do not mix VEX prefixed instructions with non-VEX instructions.

In 2012, *Advanced Vector Extensions 2* was anounced for the Haswell architecture of Intel. **AVX2** includes the expansion of many integer operations to 256 bits, support for gather/scatter operations to load/store registers from/to non-contiguous memory locations, addition of more permute instructions of elements within vector and also the FMA instruction set[2].

3.5 SSE5, XOP, FMA4

A big confusion was caused in the future directions of new instruction sets, both Intel and AMD has been proposed different instruction sets that have been changed in response to design aspects. Actually, Bulldozer AMD's processor implements the XOP and FMA4 instruction set and has compatibility with AVX instruction set. For the second generation of Bulldozer architecture, called Piledriver will support FMA3, which consist in a series of fused multiply and add operations over floating point elements. Currently the only definition of SSE5 will contain is provided by AMD in [2].

4 ARM-based processors.

ARM architecture is widely used by its low power consumption and high distribution in several devices. ARM architecture is licensable, it allows to companies to license the ARM designs to manufacture their own system on chip (SoC) to add more components. Several companies have made their own SoC's and distributed their products in mobile computing, routers and switches, tablets and network card devices.

ARM is a RISC pipelined processor with 16 32-bit registers and fixed size encoding instruction architecture. All of their instructions are conditionally executed according to condition codes in the flags register. It has a throughput of one clock cycle per instruction in most of operations. ARM has powerful index addressing modes along with a barrel shifter, which can shift a register without performance penalty in most of arithmetic and address calculation instructions.

Thumb is an alternative variable size encoding scheme released in 1994, with the purpose to improve the code density used by fixed size encodings. Thumb is able to encode a subset of ARM instructions

[2]FMA stands for *fused multiply and add* operation.

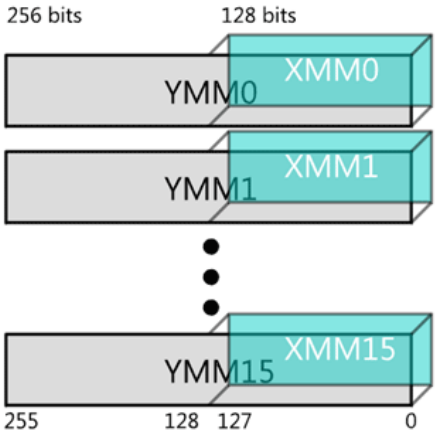

Figure 4: Register set of AVX architecture, note that `YMM` registers share bits with the `XMM` registers.

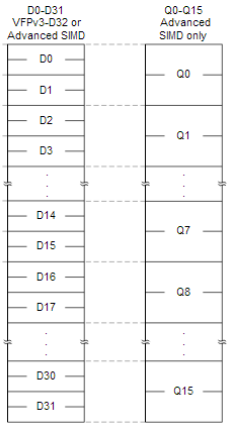

Figure 5: Register set of NEON architecture.

in 16-bit operation codes. The shorter operation codes allow to increase performance compared with 32-bit from ARM code. This optimization left out conditional execution of instructions and then *Thumb-2* technology emerged to solve this problem. *Thumb-2* can support variable size encodings and also conditional execution with the `IT` instruction.

4.1 NEON instruction set.

ARM architecture has a *Vector Floating Point* (VFP) coprocessor which extends the ARM instruction set. The VFP architecture was intended to support execution of short vector mode instructions but these operated on each vector element sequentially and thus did not offer the performance of true SIMD parallelism.

Due to VFP does not provide a real SIMD execution, ARM decided to integrate the *Advanced SIMD* extension, known as NEON or Media Processing Engine. NEON involves a 128-bit instruction set to provide acceleration of media and signal processing applications.

NEON has 16 128-bit registers to process packed SIMD operations over 8, 16 and 32-bit elements. Their registers can be accessed through vector load/store instructions, just one instruction can load or store a set of consecutive data in memory. Registers has aliases to refer them, i.e. the set `D0-D31` are 64-bit registers which also refer to `Q0-Q15` 128-bit registers, this can be seen in figure 5.

In [23] presents an application of NEON in the AVS video decoding acceleration using a Cortex-A processor. In [21, 8] shows how to take advantage of NEON unit applied to secure implementations of cryptographic protocols.

5 Implementation Aspects.

5.1 Intrinsics.

A nice feature for the use of SIMD instructions is the ability to use high level functions instead of assembler mnemonics, i.e. there are a set of C/C++ functions and data types that can be used directly in C code, then compiler translates this code into assembler SIMD instructions. This translation by compiler enables further optimizations for an specific target architecture. A large set of intrinsics supported by Intel processors can be found in [12], for AMD processors in [16] and for ARM processor with NEON unit in [6].

Compiler	Version	Optimization Flags	SIMD Intrinsics support
GNU C Compiler	4.7	-ftree-vectorize -faggressive-loop-optimizations -funsafe-loop-optimization -ftree-parallelize-loops=n	-msse4.2 -msse2avx -mavx -march=corei7-avx
Intel C Compiler	11	/Qvec-report(n) /Qguide /Qparallel	/QxSSE4.2 /QxSSE4.1 /QaxAVX
Visual Studio	2012	/Qpar /Qpar-report:12 /Qvec-report:12	/arch:[IA32—SSE—SSE2—AVX]

Table 1: Three of the most used compilers has special flags to support the use of SIMD instructions.

5.2 Compiler Auto Vectorization.

Research in compiler development and optimizations has result in better performance in applications. The inclusion of SIMD features in processor enables that compilers provide a auto vectorization of code, it means compiler automatically detects possible parallel computing over data.

Compilers found this parallelism in applications and translates into vectorized code, they achieve this with clever algorithms that receive some hints given by programmer to found such parallelism. These hints could be the use of some reserved keywords or with implicitly through the use of simple loops.

In both GNU C Compiler, Intel C Compiler and Visual Studio 2012 provide compiler flags to specify optimization level using vectorized code. In table 1 shows compiler flags used for SIMD applications.

6 Concluding remarks.

SIMD processing has been grown in the latest processor development. End user applications now demand more parallel processing through high resolution video and 3D graphics. Scientific computation can be done in modern architectures without requiring sophisticated hardware.

Intel, AMD and ARM have the most advanced instruction sets in their processors, they share some characteristics and in the future it is possible that they converge to unify sets of instructions. A high level of optimization in compilers allows to translate user code into efficient vectorized code.

Remarkably, it is possible to extract more parallelism with the use of multi core approach, each core containing a dedicated SIMD unit. In the other hand, general purpose GPU's provide a wide SIMD execution if extra hardware enters in the budget. In the near future high power processors will introduce larger registers, they will improve the throughput of operations and memory access.

References

[1] Aberdeen, D., and Baxter, J. Emmerald: A fast matrix-matrix multiply using intel's sse instructions. *Concurrency and Computation: Practice and Experience 13* (2001), 103–119.

[2] AMD. Amd64 technology 128-bit sse5 instruction set. `http://developer.amd.com/wordpress/media/2012/10/AMD64_128_Bit_SSE5_Instrs.pdf`, aug 2007.

[3] AMD. 3DNow!^{TM} Instructions are Being Deprecated. `http://developer.amd.com/community/blog/3dnow-deprecated/`, aug 2010.

[4] Atkins, M. Performance and the i860 microprocessor. *Micro, IEEE 11*, 5 (1991), 24–27.

[5] Bouknight, W. J., Denenberg, S., McIntyre, D., Randall, J. M., Sameh, A., and Slotnick, D. The illiac iv system. *Proceedings of the IEEE 60*, 4 (1972), 369–388.

[6] Compiler, G. C. Arm neon intrinsics. http://gcc.gnu.org/onlinedocs/gcc/ARM-NEON-Intrinsics.html.

[7] Internet protocol small computer system interface (iscsi) cyclic redundancy check (crc)/checksum considerations. http://www.rfc-editor.org/rfc/rfc3385.txt, sep 2002.

[8] Faz-Hernandez, A., Longa, P., and Sanchez, A. H. Efficient and secure algorithms for glv-based scalar multiplication and their implementation on glv-gls curves. Cryptology ePrint Archive, Report 2013/158, 2013. http://eprint.iacr.org/.

[9] Flynn, M. Very high-speed computing systems. *Proceedings of the IEEE 54*, 12 (1966), 1901–1909.

[10] Haque, I. S., Pande, V. S., and Walters, W. P. Anatomy of high-performance 2d similarity calculations. *Journal of Chemical Information and Modeling 51*, 9 (2011), 2345–2351.

[11] Intel. Hardware design site archives. intel® pentium processor with mmx™ technology documentation. http://www.intel.com/design/archives/Processors/mmx/.

[12] Intel. Intel® intrinsics guide. http://software.intel.com/en-us/articles/intel-intrinsics-guide.

[13] Intel. Intel® sse4 programming reference. http://software.intel.com/sites/default/files/m/9/4/2/d/5/17971-intel_20sse4_20programming_20reference.pdf.

[14] Intel. Pentium® processor with mmx™ technology performance brief. http://download.intel.com/design/archives/processors/mmx/docs/24328605.pdf.

[15] Kohn, L., and Margulis, N. Introducing the intel i860 64-bit microprocessor. *Micro, IEEE 9*, 4 (1989), 15–30.

[16] MSDN, M. 3dnow! intrinsics. http://msdn.microsoft.com/en-us/library/k4s9ed2x(v=vs.100).aspx.

[17] Oberman, S., Favor, G., and Weber, F. Amd 3dnow! technology: architecture and implementations. *Micro, IEEE 19*, 2 (1999), 37–48.

[18] Patterson, D. A., and Ditzel, D. R. The case for the reduced instruction set computer. *SIGARCH Comput. Archit. News 8*, 6 (Oct. 1980), 25–33.

[19] Purcell, C. J. The control data star-100: performance measurements. In *Proceedings of the May 6-10, 1974, national computer conference and exposition* (New York, NY, USA, 1974), AFIPS '74, ACM, pp. 385–387.

[20] Russell, R. M. The cray-1 computer system. *Commun. ACM 21*, 1 (Jan. 1978), 63–72.

[21] Sánchez, A., and Rodríguez-Henríquez, F. NEON implementation of an attribute-based encryption scheme. In *Technical Report CACR 2013-07* (2013). Available at: http://cacr.uwaterloo.ca/techreports/2013/cacr2013-07.pdf.

[22] Takahashi, A., Soliman, M., and Sedukhin, S. Parallel lu-decomposition on pentium streaming simd extensions. In *High Performance Computing*, A. Veidenbaum, K. Joe, H. Amano, and H. Aiso, Eds., vol. 2858 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003, pp. 423–430.

[23] Wan, J., Wang, R., Lv, H., Zhang, L., Wang, W., Gu, C., Zheng, Q., and Gao, W. Avs video decoding acceleration on arm cortex-a with neon. In *Signal Processing, Communication and Computing (ICSPCC), 2012 IEEE International Conference on* (2012), pp. 290–294.

[24] Watson, W. J. The ti asc: a highly modular and flexible super computer architecture. In *Proceedings of the December 5-7, 1972, fall joint computer conference, part I* (New York, NY, USA, 1972), AFIPS '72 (Fall, part I), ACM, pp. 221–228.