A Fast New DES Implementation in Software

Eli Biham

Computer Science Department
Technion – Israel Institute of Technology
Haifa 32000, Israel
Email: biham@cs.technion.ac.il
WWW: http://www.cs.technion.ac.il/~biham/

Abstract. In this paper we describe a fast new DES implementation. This implementation is about five times faster than the fastest known DES implementation on a (64-bit) Alpha computer, and about three times faster than than our new optimized DES implementation on 64-bit computers. This implementation uses a non-standard representation, and view the processor as a SIMD computer, i.e., as 64 parallel one-bit processors computing the same instruction. We also discuss the application of this implementation to other ciphers. We describe a new optimized standard implementation of DES on 64-bit processors, which is about twice faster than the fastest known standard DES implementation on the same processor. Our implementations can also be used for fast exhaustive search in software, which can find a key in only a few days or a few weeks on existing parallel computers and computer networks.

1 Introduction

In this paper we describe a new implementation of DES[4], which can be very efficiently executed in software. This implementation is best used with a non-standard order of the bits of the DES blocks. This implementation does not suffer from high overhead of computing permutations of bits. Instead, we view a processor with (for example) 64-bit words, as a SIMD parallel computer which can compute 64 one-bit operations simultaneously, while the 64-bits of each block are set in 64 different words (of which the first bit is always of the first block, the second bit belongs to the second block, etc.).

The operations that DES uses are as follows: The XOR operation: in our view the XOR operation of the processor computes 64 one-bit XORs. The expansion and permutation operations: these operations do not cost any operation, since instead of changing the order of words (or duplicating words), we can address the required word directly. We remain with the S boxes. Usual implementations of S boxes use table lookups. However, in our representation, table lookups are very inefficient, since we have to collect six bits, each bit from a different word,

Cipher	Speed
DES (Eric Young's libdes)	28
Gost	8*
SAFER	22*
Blowfish	34*
Our DES Implementation	46
Our DES Implementation – triple DES	22
Our fastest DES	137
Our fastest DES – Triple DES	46

* Estimation, based on [9].

Table 1. The speeds of our implementations and of various ciphers on a 300MHz Alpha 8400 processor (in Mbps).

combine them into one index to the table, and after the table lookup take the four resultant bits and put each of them in a different word.

We observed that there is a much faster implementation of the S boxes in our representation: they can be represented by their logical gate circuit. In such an implementation each S box is typically represented by about 100 gates, and thus we can implement an S box by about 100 instructions.

We actually view the whole cipher by its gate circuit, and apply it in software. In this implementation we actually compute the circuit 64 times in parallel (as the size of the processor word), and thus can gain a high speedup even though we use very simple operations. In average, on 64-bit processors, each S box costs about 1.5 instructions for each encrypted block, while each instruction takes only one clock cycle.

The full circuit of DES contains about 16000 gates (including the key scheduling, which costs nothing), and thus we can compute DES 64 times in about 16000 instructions on 64-bit processors. In average we result with about 260 instructions for the encryption of each DES block. Conversion from and to the standard block representation takes (together) about 40 instructions per block, and thus encryption of standard representations with our implementation takes about 300 instructions. For comparison, our fast standard implementation of DES, described in this paper, requires about 634 instructions for each block.

Table 1 summarizes the speeds of our implementations, a standard fast DES implementation (Eric Young's libdes), and of various fast ciphers.

The same idea can be applied to other ciphers. Our implementation of these ciphers is efficient especially when the cipher does not use all the power of the machine instructions (i.e., when each instruction mixes only a few of the bits, such as S boxes or eight-bit additions on 32-bit processors), and when the word size of the processor is large (such as 64 bits, when the cipher use shorter registers). For example, our implementation of Feal[11] is expected to be about 2.5–5 times faster than direct implementations. Both variants of Lucifer[1,12] and GOST[10] can also be applied very efficiently using this implementation. Our implementation of ciphers which use more complex operations (such as multiplication, or large S boxes) requires more instructions to simulate the complex operations, and is thus less efficient.

In Section 3 we describe an optimized standard implementation on 64-bit computers. It uses the 64-bit registers of a 64-bit processor, and runs almost twice faster than the fastest implementation (designed for 32-bit architectures) on the same processor. It even runs faster than fast ciphers such as GOST[10], SAFER[2], and Blowfish[10]. The speed is gained by using the long 64-bit registers effectively — by all other means this is a standard implementation. We suggest a new DES-like cipher, to which we call *WDES*, based on the structure of this fast implementation, but is about 2.5 times faster.

In Section 4 we discuss using these fast implementations for exhaustive search, and conclude that it is applicable even today using existing general purpose parallel computers and computer networks.

2 The New Non-Standard DES Implementation

This implementation uses a non-standard representation of the data in software, and in particular it does not have any table lookup. Instead of encrypting many 64-bit words, one at a time, we encrypt simultaneously 64 words, and each operation encrypts one bit in each of the 64 words.

Actually, we view a 64-bit processor as a SIMD computer with 64 one-bit processors. This implementation simulates a fast DES hardware whose number of gates is minimal, and computes each gate by a single instruction. In particular, the S boxes are computed by their gate-circuit, using the XOR, AND, OR, and NOT operations, and the permutations and expansions do not require any instruction, since they can be viewed as only changing the naming of the registers. Although the S boxes are implemented in more instructions than in usual implementations, the parallelism of this implementation speeds up the implementation much more than the S box implementation reduces it. Moreover, some of the operations can be optimized out in some cases, such as if some parts of the S boxes are similar (same or complement).

We represent the S boxes by their gate circuit using the best-known XOR, AND, OR and NOT operations, optimized to reduce the total number of gates. Although the problem of finding the best such circuit is still open, we found the following optimization which requires at most 132 gates per DES S box, and only 100 gates in average. In the description we denote the six input bits by

	Instructions
Expansion	0
Key mixing	48
P	0
XOR with the left half	32
S boxes	$8 \cdot 100 = 800$ (in average)
load+store	$8 \cdot (6 + 6\text{load} + 4\text{load} + 4\text{store}) = 160$
Total per round:	1040

Table 2. The number of instructions in each round on Alpha.

	Total	Average per Block
IP,FP	0	0
16 rounds:	$16 \cdot 1040 = 16640$	260
		4 gates per bit
Conversion of representation	2500	40

Table 3. The number of instructions in DES on Alpha.

$abcdef$. We compute all the 16 functions of d and e into 14 registers (excluding the constant 0 or constant 1). It requires two NOTs ($\bar{d}, \bar{e}$) and 10 additional operations ($0, 1, d, e, \bar{d}, \bar{e}$ are already known). This computation is done only once for each S box. For each output bit of the S box we compute the result using these functions. We use six operations for each line of the S box and six operations to combine the results, together 30 operations for each output bit. In total we use at most $12 + 4 \cdot 30 = 132$ gates for each S box, but in average we need only about 100 gates per S box. Each combination of four values (the four values of b, c or the four values of a, f, e.g., combining the quarters of each of the four lines $((b = c = 0), (b = 0, c = 1), (b = 1, c = 0), (b = c = 1))$, or combining the four lines) are combined by (assuming the first case):

$$\left(\underline{f_{00}} \oplus c \cdot \underline{(f_{00} \oplus f_{01})}\right) \oplus b \cdot \left(\underline{(f_{00} \oplus f_{10})} \oplus c \cdot \underline{(f_{00} \oplus f_{01} \oplus f_{10} \oplus f_{11})}\right),$$

where the underlined values are known constants, and $f_{bc} = S(abcdef)$, where d, e are the actual values of the input (f_{bc} is one of the 16 values kept in registers above), and a, f are the values assumed for a, f, to be instantiated in the next step. More accurately, in the intermediate steps we compute the combinations of S box entries as suggested by the above equation (e.g., f_{00}, $f_{00} \oplus f_{01}$, $f_{00} \oplus f_{10}$, $f_{00} \oplus f_{01} \oplus f_{10} \oplus f_{11}$), rather than the various values of the entries themselves.

Tables 2 and 3 describe the maximum number of gates per round and for the full DES. Therefore, we expect the speed to be about $300 \cdot 2^{20}/4 = 75$Mbps on

300MHz Alpha processors. In practice, we achieve speeds of about 137Mbps, since the processor can apply more than one instruction in each clock cycle.

Conversion between the standard and the non-standard representations can also be done in about 1250 instructions. Doing this twice, before and after encryption, takes about 2500 instructions, which are about 40 instructions for each encrypted block.

This implementation can actually be applied to any cipher, but the efficiency of the implementation depends on many factors, such as the efficiency of the original cipher, the word size of the processor, and the complexity of the operations that the cipher uses. The implementation is especially attractive to ciphers whose operations are simple (no multiplication for example), use only small S boxes (thus their gate complexity is small), or use small register sizes (thus cannot use the full power of modern processors). Examples of such ciphers are Lucifer[1,12], GOST[10] and Feal[11].

In the case of Feal, standard implementations require about 22 instructions for each application of the round function (4 loads, 2 load + 2 XORs for key mixing, 2 for XOR, 2 additions (S_0), 2 additions with carry (S_1; each might take two operations), 4 rotations and 4 XORS to mix with the left half of the data). The right-round cipher takes thus about $8 \cdot 22 = 176$ instructions (not counting the initial and final key mixing which can take a few additional instructions).

Our implementation requires 34 or 35 instructions for an eight-bit addition (one or two for the LSB, depends whether this is S_0 or S_1, 1 for the carry and 2 for the second bit. We need three additional instructions for computing each additional carry and two XORs for each additional bit: In total we need $1 + (1 + 2) + 6 \cdot (2 + 3) = 34$ instructions for S_0 and 35 for S_1). In total the F function requires $16 + 16 + 16 + 35 + 34 + 35 + 34 + 32 + 32 + 32 + 8 + 8 = 298$ instructions (16 XORs, 16 key loads+mixings, four S boxes, 32 loads, 32 stores, 32 mixings with the left half, and 8+8 extra loads+stores if necessary). The eight round Feal can then be implemented in $8 \cdot 298 + 64 + 64 = 2512$ instructions (64 for each of the initial and final key mixings). In average we get that only about $2512/64 = 39$ instructions per block, which is more than four times faster than standard implementations. Even if we do the conversion from/to the standard representation (which costs 40 instructions per block), our implementation takes only about $39 + 40 = 79$ instructions, which is more than twice faster than the standard implementations.

Both variants of Lucifer[1,12] and GOST[10] can also be applied very efficiently using this implementation.

This implementation can be used for fast encryption and decryption, using the same key in all the 64 encryptions (i.e., the key words contain only 0 or -1), or for exhaustive search using the same plaintexts but different keys. We can also use different plaintexts with different keys, if it is of an advantage to the application.

This implementation can be used in three ways:

1. Encryption/decryption in standard representations, compatible to other DES implementations.
2. Encryption/decryption of large blocks, such as of disk clusters, or large communication packets. In this case, it is not important to use the standard representation, and thus our implementation is even faster, since conversion should not be done.
3. Application to exhaustive search.

It is easy to see that applications of this implementation in the ECB mode is very fast, but as usual in ECB modes, it suffers from many disadvantages. It would be preferable to use standard CBC, CFB and OFB modes with this implementation, but this is impossible due to their sequential order. However, it is possible to use this implementation for standard CBC decryption, since the whole data can be decrypted in parallel, and then each result can be mixed with the previous ciphertext. It is also possible to apply CFB decryption in a similar way. Therefore, this implementation can be used for fast decryption in standard modes, even when encryption is done by usual standard implementations.

64 parallel CBC encryption modes can be applied in this implementation by choosing 64 initial values for the 64 block encrypted simultaneously, and apply CBC on the full $64^2 = 4096$-bit blocks. In this case we can also encrypt under a different key in each of the 64 parallel CBC modes — it might be especially attractive when a server has to encrypt data to many clients in parallel.

This implementation is even faster when conversion from/to standard representation is not applied. In this case, DES is applied, but with a non-standard order of the plaintext/ciphertext bits. To protect against multiple occurrence of the same plaintexts (actually the 64 bits that enter one real DES in the non-standard representation) we should use new modes.

The ECB mode of this implementation takes the 4096 bits of the data, and encrypts them as is. A CBC-like mode can have an initial value of 4096 bits (which can be derived from a 64-bit value), and apply CBC on the 4096-bit cipher. This mode actually applies 64 standard CBC modes in parallel, one for each of the DES applications in the non-standard representation. An improvement of this mode can mix the bits of each register, for example by rotating register i (containing the i'th bits of the standard blocks) by i bits after adding i to the value of the register. A CFB-like and OFB-like modes can be designed in a similar way.

	Operations	Number of Instructions
key XOR	1 load+XOR	2
EPS	8 table lookups	$8 \cdot 3 = 24$ (extbl, add, lookup)
XORing L with the S boxes	8 XORs	8
Total		34

Table 4. The number of instructions in each round on Alpha.

	Operations	Number of Instructions
IP	5 times (3 XORs, 2 shifts, 1 AND)	$5 \cdot 6 = 30$
E	Initial Expansion	26
16 rounds	each 34 instructions	$16 \cdot 34 = 544$
	Removal of expansion	4
FP	Final permutation	30
Total		634

Table 5. The number of instructions in DES on Alpha.

3 A Fast Standard DES Implementation on 64-bit Processors

DES can be applied very efficiently on 64-bit processors. Unlike on 32-bit processors, on 64-bit processors, the right half expanded to 48 bits can be stored in one word. Moreover, by substituting every group of six bits entering into the S boxes in a separate byte, we can directly access the S box table by referencing via a single byte.

We apply the initial and final permutations by lookup tables from each byte to 64-bits, and XORing the results of the various table lookups.

We apply each round by XORing the right half (represented as eight bytes, in each six bits are used) by a subkey (represented in the same way). Then, eight table lookups apply the eight S boxes, and the results are XORed. Each S box already includes the P permutation and the E expansion in its 64-bit result. Note that due to this representation, several (duplicated) bits of the two halves should be omitted by the final permutation.

Tables 4 and 5 describe the number of operations required by this implementation, with the number of instructions on an Alpha processor. We implemented this code in C on a 300MHz Alpha and got encryption speed of 46Mbps. Triple DES runs at 22Mbps (since some IP, FP's can be discarded). On the same processor, Eric Young's libdes (single-DES) runs at 28Mbps.

Some comments on this implementation:

1. The eight S boxes are applied in parallel, and thus pipelining can use it without pipeline stalls. In other ciphers and hash functions, like Feal[11], Khufu[3], Khafre[3], and MD4[7], MD5[8], SHA-1[5,6], each operation depends on the output of the previous operation, and thus might result with pipeline stalls, especially on newer or future processors which can compute several instructions simultaneously.
2. All the tables and the variables take together about 4Kbytes, and enter easily into the cache.
3. Still in DES the input of the next round depends on the output of the preceding one. Although in practice this does not slow the execution, we have another solution. In DES, the input of each S box depends only on the output of only six S boxes in the previous round. Thus, the code can be optimized to start computing the next round while still computing the preceding one. This can speed up implementations on pipelined processors, where we can compute several instances in parallel.
4. Unlike some (although not all) DES implementations, we implement each S box as one table lookup, rather than combining pairs of S boxes into one lookup. The latter is more than twice slower, since the tables become larger than the size of the on-chip cache[1].

3.1 WDES

We can use this fast code to design a new, even faster, and more secure cipher, to which we call *WDES*. We convert the code by removing IP, FP, and changing the EPS operations (S boxes followed by P followed by E, as used in this implementation) into S boxes from 8 bits to 64 bits. These S boxes can be much better than the original, since each S box affects *all* the bits of *all* the S boxes in the next round (rather than one bit in only six S boxes).

WDES has 128-bit blocks, and it runs much faster than DES, with the same number of rounds (since the blocksize is larger, and the slow initial and final permutations are discarded): its speed is 106Mbps on the same processor as in Table 1.

4 Exhaustive Search on Powerful Computers and Networks

In this section we study the possibilities of exhaustive search on several kinds of machines and networks. We assume using the fast implementation described in

[1] On Pentium, however, the latter is twice faster using the same C code

the previous section.

Note that results similar to the ones described here hold also for breaking UNIX passwords, which are chosen from up to eight printable characters. In this case the password space has 96^8 passwords, while each password trial requires 25 encryptions (the salt should not be taken into account, since it is known to the attacker, and the encryption code can be justified to the specific value of the salt). Therefore, about $25 \cdot 96^8 \approx 2^{57}$ passwords should be tried, or about 2^{56} in average.

4.1 Special Purpose Computers

We can build a special purpose computer with very long registers, without the expensive operations (such as multiplication and floating point operations), and only with simple instructions, such as XOR, AND, OR, NOT. Assume that in a Pentium processor we remove the expensive operations, and use the extra chip space to increase the size of the registers to 1000 bits. Then, we need only 150 processors to search the keys exhaustively in one year in average (or six months in average using the attack based on the complementation property).

It is possible theoretically to build a machine with million-bit registers. Unexpectedly, we now know that such a machine was actually built with the support of the NSA: Cray Computers had announced in March 1995 about such a computer that can apply 2^{45} bit operations every second on a million one-bit processors (see Figure 1). This computer can compute 2^{45} bit-operations every second, and thus can compute about $2^{45}/16000 = 2^{45}/2^{14} = 2^{31}$ DES encryptions every second. Therefore, we can apply the searches on this machine with the following results:

Search of	Time	Notes
40 bits	512 sec=8.5 min, 4.25 min in av.	Exportable ciphers
43 bits	4096 sec=an hour, 1/2 an hour in av.	Linear Cryptanalysis
47 bits	2^{16} sec=a day, 12 hours in av.	Differential Cryptanalysis
56 bits	2^{25} sec=a year, 1/2 an year in av.	Full key search

Cray Computers has bankrupted, since nobody had bought this computer. Probably the NSA had a faster machine.

4.2 General Purpose Parallel Computers

It is known that Sandia National Labs has a parallel computer of 9000 200MHz Pentium-Pro processors. This parallel computer can compute about $9000 \cdot 200 \cdot 2^{20} \cdot 32 = 2^{46}$ bit operations every second. Thus, it can compute about $2^{46}/16000 =$

$2^{46}/2^{14} = 2^{32}$ DES encryptions in each second. Therefore, we can apply the searches on this machine with the following results:

Search of	Time	Notes
40 bits	256 sec=4 min, 2 min in av.	Exportable ciphers
43 bits	2048 sec=1/2 an hour, 15 min in av.	Linear Cryptanalysis
47 bits	2^{15} sec=12 hours, 6 hours in av.	Differential Cryptanalysis
56 bits	2^{24} sec=6 months, 3 months in av.	Full key search

When we apply the attack using the complementation property, exhaustive search of the full key space takes in average only about six weeks.

4.3 Internet and the DES Worm

We can use the Internet for our exhaustive search, just as RSA factorization teams are doing. Assume that an average computer on the Internet is a single 32-bit 133MHz RISC processor. Such a processor can encrypt about 2^{18} blocks every second. Therefore,

– Searching 40 bits takes about $2^{40}/2^{18}/2 = 2^{21}$ seconds in average, which are about three weeks on a single processor. 1000 computers can do it in about half an hours.
– Searching 43 bits takes about six months in average. 1000 computers can do it in six hours.
– Searching 47 bits takes about 8 years in average on a single processor. 1000 computers can do it in four days, and 4000 computers can do it in one day.
– Searching all the 56 bits takes about 4000 years in average on a single processor. 4000 computers can do it in a year (or in six months using the complementation property). It is practical to have this number of computers participating legally over the Internet: this is about the same number of computers as the RSA factorizations use. Million computers can do it in two days in average (or in one day using the complementation property).

At this point it is possible in practice to achieve participation of several thousands computers legally over the Internet. However, it is simpler, and faster to do it illegally[2]. A worm, for which we call the *DES worm*, can break into many computers over the Internet, and use their idle cycles for exhaustive search. The worm verifies that only one copy of it is executed on each computer (of course on computers with several processors it can execute several copies to increase performance). The DES worm makes sure it cannot be easily noticed: it does not

[2] The Author does not recommend to do it, but we should always be aware that such a threat exists.

need much memory anyway, and it is executed at the lowest possible priority, so it does not disturb other applications on the same computer.

If the DES worm can get hold of about a million computers over the Internet, and assuming that it get at least half of their cycles (people are usually not working over nights), the DES worm can find a key in four days in average (or in two days using the complementation property). Moreover, since most computers over the Internet are not used in weekends (which last over 60 hours from Friday evening to Monday morning), the DES worm can use all the cycles and find a key in one weekend.

5 Acknowledgements

We are grateful to Adi Shamir, Ross Anderson and the referees for their various remarks and suggestions that improved the results and exposition of this paper. Some of this work has been done while the author was visiting the computer laboratory at the university of Cambridge, and in particular using their Alpha computer. This research was supported by the fund for the promotion of research at the Technion.

References

1. H. Feistel, *Cryptography and Data Security*, Scientific American, Vol. 228, No. 5, pp. 15–23, May 1973.
2. James L. Massey, *SAFER-K64: A Byte Oriented Block Ciphering Algorithm*, proceedings of Fast Software Encryption, Cambridge, Lecture Notes in Computer Science, pp. 1–17, 1993.
3. Ralph C. Merkle, *Fast Software Encryption Functions*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of CRYPTO'90, pp. 476–501, 1990.
4. National Bureau of Standards, *Data Encryption Standard*, U.S. Department of Commerce, FIPS pub. 46, January 1977.
5. National Institute of Standard Technology, *Secure Hash Standard*, U.S. Department of Commerce, FIPS pub. 180, May 1993.
6. National Institute of Standard Technology, *Secure Hash Standard*, U.S. Department of Commerce, FIPS pub. 180-1, April 1995.
7. Ronald L. Rivest, *The MD4 Message Digest Algorithm*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of CRYPTO'90, pp. 303-311, 1990.
8. Ronald L. Rivest, *The MD5 Message Digest Algorithm*, Internet Request for Comments, RFC 1321, April 1992.
9. Michael Roe, *Performence of Block Ciphers and Hash Functions – One Year Later*, proceedings of Fast Software Encryption, Leuven, Lecture Notes in Computer Science, pp. 359–362, 1994.
10. Bruce Schneier, *Applied Cryptography, Protocols, Algorithms, and Source Code in C*, second edition, John Willey & Sons, 1996.

11. Akihiro Shimizu, Shoji Miyaguchi, *Fast Data Encryption Algorithm FEAL*, Lecture Notes in Computer Science, Advances in Cryptology, proceedings of EUROCRYPT'87, pp. 267–278, 1987.
12. Arthur Sorkin, *Lucifer, a Cryptographic Algorithm*, Cryptologia, Vol. 8, No. 1, pp. 22–41, January 1984.

OTC 03/07 1942 CRAY COMPUTER CORP. COMPLETES INITIAL TESTING

COLORADO SPRINGS, Colo., March 7 /PRNewswire/ – Cray Computer Corp. (Nasdaq: CRAY) reported today the successful test and demonstration on March 2, 1995, of an array of 256,000 single bit processors packaged using the company's multi-chip-module technology. This array is a major technical component of the CRAY-3/Super Scalable System (CRAY-3/SSS) that is being **jointly developed by the company, the National Security Agency** and the Supercomputing Research Center (SRC) which was originally announced on August 17, 1994. This test and demonstration completes the first of a number of major tasks required under the Development Contract.

Researchers from the SRC verified correctness of operation of the 256, 000 single bit processor array (approximately 4,000 individual Integrated Circuits), which is the first half of a 512,000 singe bit processor array called for in the development contract. This array is coupled to a CRAY-3. The CRAY-3/SSS utilizes the Processor-In-Memory (PIM) chips, developed by the SRC. Both **NSA** and SRC are **providing significant technical assistance** in both the software and hardware aspects of the system.

Once completed, the high performance system will consist of a dual processor 2,048 million byte CRAY-3 and a 512,000 single bit processor Single Instruction Multiple Data (SIMD) array with a 128 million byte memory. This CRAY-3/Super Scalable System will provide high-performance vector parallel processing, scalable parallel processing and the combination of both in a hybrid mode featuring extremely high bandwidth between the PIM processor array and the CRAY-3. The current schedule for completion of the Development Contract is the end of July 1995 including a 90 day public Internet access demonstration.

For suitable applications, a SIMD processor array of 1 million processors would provide up to 32 Trillion Bit Operations per Second and price/performance unavailable today on any other high-performance platform. The CRAY-3 system with the SSS option will be offered as an application specific product. The joint development contract is part of the Federal Government's High Performance Computing and Communications program.

Charles Breckenridge, executive vice president of Marketing at Cray Computer Corp. said, "The CRAY-3/SSS will provide unparalleled performance for many promising applications. We are pleased to participate in this **transfer of government technology**, and we are eager to help potential customers explore and develop appropriate applications."

Cray Computer Corp. is engaged in the design, development, manufacture and marketing of the CRAY-3, CRAY-3/SSS and CRAY-4 high- performance computer systems.

CONTACT: Charles Breckenridge, executive VP of Marketing, or Terry Willkom, president, of Cray Computer, 719-579-6464; or David Gould of Chip Shots Inc., 408-541-8706

Fig. 1. Cray Computer Corp. press release of March 1995.