Performance and Overhead in a Hybrid Reconfigurable Computer

Osman Devrim Fidanci[1], Dan Poznanovic[2], Kris Gaj[3], Tarek El-Ghazawi[1], Nikitas Alexandridis[1]
[1]George Washington University, [2]SRC Computers Inc., [3]George Mason University

Abstract

In this paper, we overview general hardware architecture and a programming model of SRC-6E™ reconfigurable computers, and compare the performance of the SRC-6E machine vs. Intel® Pentium IV™. SRC-6E execution time measurements have been performed using three different approaches. In the first approach, the entire end-to-end execution time is taken into account. In the second approach, the configuration time of FPGAs have been omitted. In the third approach both configuration and data transfer overheads have been omitted. All measurements have been done for different numbers of data blocks. The results show that the SRC-6E can outperform a general-purpose microprocessor for computationally intensive algorithms by a factor of over 1500. However, overhead due to configuration and data transfer must be properly dealt with by the application or the system's run-time environment to achieve the full throughput potential. Some techniques are suggested to minimize the influence of the configuration time and maximize the overall end-to-end system performance[1].

1: Introduction

The SRC-6E Reconfigurable Computing Environment is one of the first general-purpose reconfigurable computing machines combining the flexibility of traditional microprocessors with the power of Field Programmable Gate Arrays (FPGAs). In this environment, computations can be divided into those executed using microprocessor instructions, and those executed in reconfigurable hardware. The programming model is aimed at separating programmers from the details of the hardware description, and allowing them to focus on an implemented function. This approach allows the use of software programmers and mathematicians in the development of the code, and substantially decreases the time to the solution.

In this paper we investigate the possible speed-up that can be obtained using the SRC-6E Reconfigurable Computing Environment vs. a traditional PC based on the Pentium 4 microprocessor. Our benchmarks consist of the high-throughput implementations of Triple DES and DES Breaker algorithms in both environments. Triple DES, is one of the three standardized secret-key ciphers recommended for use in the U.S. government, and is widely used worldwide in multiple commercial applications. DES Breaker is a technique for breaking an old encryption standard, DES, based upon an exhaustive key search algorithm, i.e., testing all possible encryption keys one by one.

2: SRC-6E General Purpose Reconfigurable Computer

2.1. Hardware architecture

SRC-6E is a hybrid-architecture platform, which consists of two double-processor boards and one Multi-Adaptive Processor (MAP™) module (see Figure 1). The MAP module consists of two MAP processors, each including two user programmable Xilinx® Virtex II XC2V™6000 FPGA devices. This way, the SRC-6E system achieves a 1:1 microprocessor to FPGA ratio. Processor boards are connected to the MAP processors through the so-called SNAP cards. A SNAP card plugs into a DIMM slot on a microprocessor motherboard and provides interconnect between the MAP board and one of the microprocessor boards. Each SNAP card can support the peak bandwidth of 800 MB/s. [1].

2.2. Programming model

The SRC-6E has a similar programming model as a conventional microprocessor-based computing system, but needs to support additional tasks in order to produce logic for the MAP reconfigurable processor, as shown in Figure 2.

There are two types of the application source files to be compiled. Source files of the first type are compiled targeting execution on the Intel platform. Source files of

---

[1] This work was partially supported by Department of Defense through the LUCITE contract no. MDA904-98-CA0810000.

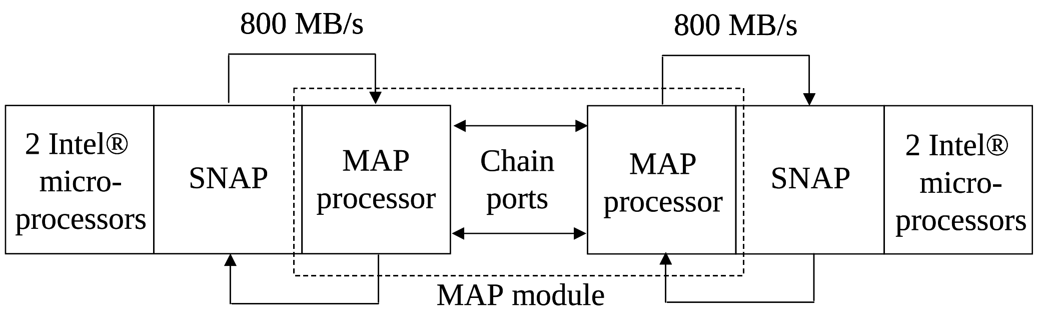

Figure 1. General Hardware Architecture of SRC-6E

the second type are compiled targeting execution on the MAP.

A file that contains the main program to be executed on the Intel processor is compiled using the microprocessor compiler to produce a relocatable object (.o) file. All files containing routines that execute on the MAP are compiled by the MAP FORTRAN compiler, mftn, or the MAP C compiler, mcc. These compilers produce several relocatable object files (.o), corresponding to respective subroutines.

Object files resulting from both the Intel® and MAP compilation steps are then linked with the MAP libraries into a single executable file. The resulting binary file may then be executed on the SRC-6E Intel and MAP hardware, or run in the emulation mode. Environment variables determine the mode of execution.

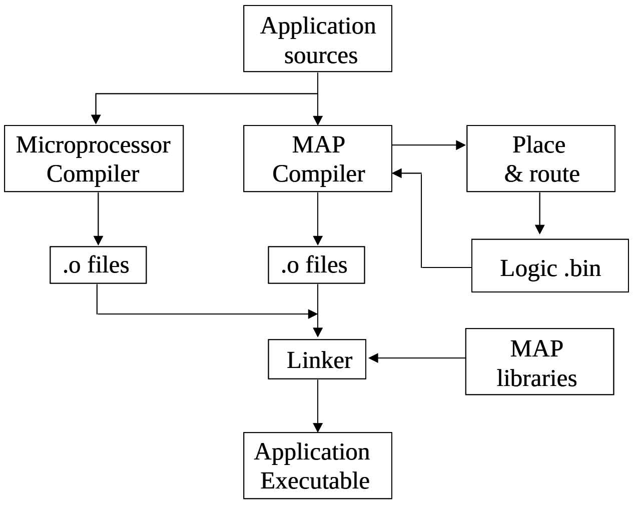

Figure 2. SRC-6E Compilation Process

2.2.1. Compiler architecture of MAP. The MAP compiler translates program sources that have been developed for the MAP execution into relocatable object files. The translation process has several steps, each performed by a distinct component of the MAP compiler, as shown in Figure 3.

The optimization phase of the compiler performs language syntax and semantic analysis followed by the classical scalar and loop optimization. During the Data Flow Graph (DFG) generation phase, the dataflow graph representing relationships between basic blocks of the program procedure is created. In this graph, basic operations are represented as nodes connected by the input and output arguments. Additional nodes are inserted for connecting blocks of graph and communicating data between blocks. Redundant nodes are pruned or optimized away [1].

The Verilog generator phase of compilation can be regarded as the "code generator" for the MAP. The Verilog generator translates the dataflow graph into its own internal format. After this translation, Verilog generator produces synthesizable Verilog code.

A commercial tool Synplify Pro^{TM} is used for the logic synthesis of the obtained Verilog file, and produces at the output the netlist EDIF file and constraint file. These files together with earlier synthesized macro files become an input for the place and route tools. The place and route tools, Xilinx® Integrated Software Environment^{TM}, complete the bitstream creation process for the MAP.

The configuration integrator is a small program that takes as input FPGA bitstream files and loads them into static structures contained in C functions. C files obtained from the MAP compilation process are then compiled together with the remaining application source files. This separate compilation of all C files is done with the Intel µP as a target microprocessor and produces as output an Intel executable. This executable can then be run on the SRC-6E system.

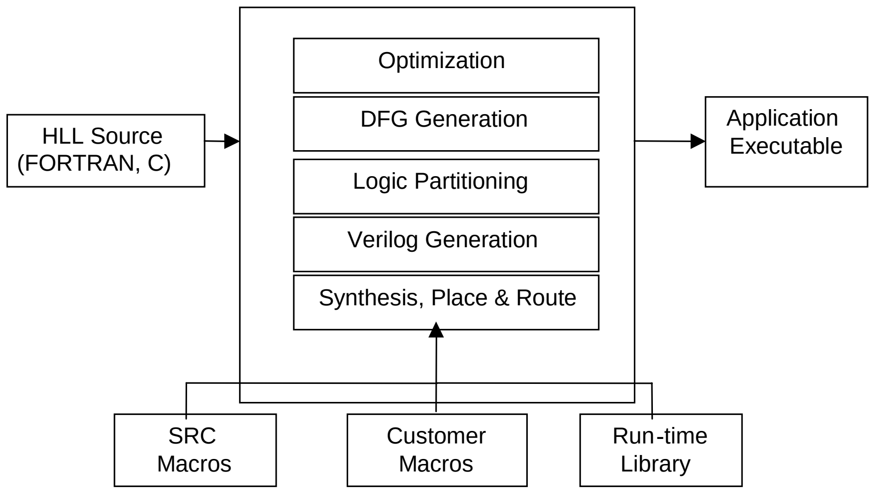

Figure 3. MAP Compilation Process

2.2.2. Macro integration. The MAP compiler translates the source code's various basic operations into macro instantiations. Here, macro can be defined as a piece of hardware logic designed to implement a certain function. Since users often wish to extend the built-in set of operators, the compiler allows users to integrate their own macros into the compilation process. The macro is invoked from within the FORTRAN subroutine or C function by means of a subroutine call.

In SRC-6E platform, macros can be categorized by various criteria, and the compiler treats them in different ways based on their characteristics. In the MAP compiler, four characteristics are particularly relevant:

A macro is "stateful" if the results it computes are dependent upon previous data it has computed or seen. In contrast "non-stateful" macro computes values using only its current inputs; it has no memory of its past values [4]. A macro is "external" if it interacts with parts of the system beyond the code block in which it lives [4].

A "pipelined" macro is able to accept new data values on its inputs in every clock cycle. Since the MAP compiler produces pipelined inner loops, the macros that will be used in such loops must be capable of pipelined operation [4].

3: Triple DES macro integration

3.1. Triple DES algorithm

In order to compare the performance of SRC-6E Reconfigurable Computing Environment and a conventional computer based on Intel® Pentium 4 processor, we have implemented the same algorithm in both environments. Our algorithm of choice is Triple DES, an American encryption standard and one of the most popular encryption algorithms used worldwide.

Triple DES by itself can be defined in a number of ways. In this paper, we use a Triple DES version proposed by Tuchman that uses only two different keys [3]. This version follows an encryption-decryption-encryption (EDE) sequence:

$$C = E_{K1}[D_{K2}[E_{K1}[P]]],$$

where E and D denote DES encryption and description, respectively. Although there is no cryptographic benefit to using decryption in the second stage, nevertheless, it provides users of Triple DES with flexibility of communicating with users of an older encryption standard - single DES. This reduction can be accomplished by setting both keys of Triple DES to the same value, as shown below:

$$C = E_{K1}[D_{K1}[E_{K1}[P]]] = E_{K1}[P]$$

Triple DES with two keys is stronger and more reliable alternative to single DES. Triple DES is used in very popular Internet applications and protocols such as PGP and S/MIME. Triple DES has also been adopted for use in the key management standards ANSI X9.17 and ISO 8732.

3.2. DES encryption and decryption structure

DES encryption takes 64-bit plaintext block (data) and 64-bit key (including 8 bits of parity) as inputs and generates a 64-bit ciphertext block (encrypted data). As shown in Figure 4, DES consists of 16 identical rounds supplemented by a few auxiliary transformations.

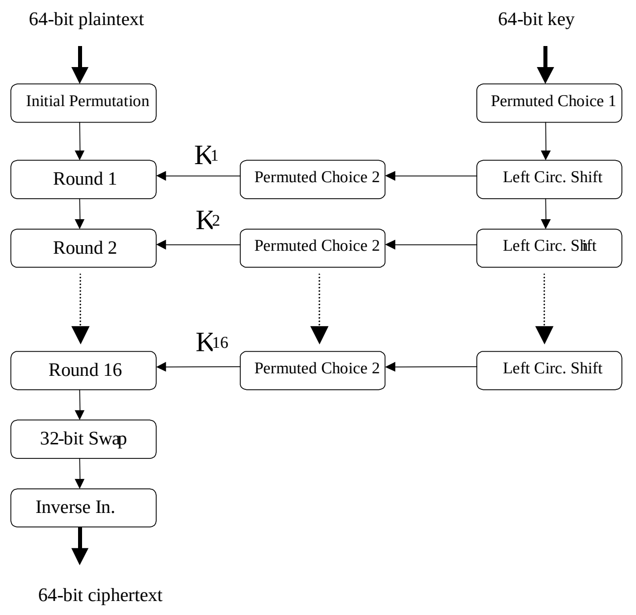

Figure 4. General architecture of DES

We have implemented DES using Verilog HDL as a non-stateful, pipelined macro with 17 pipeline stages. Triple DES was implemented in software by instantiating DES macro three times within the program subroutine.

4: Execution time measurements for the Triple DES application

4.1. SRC-6E MAP measurements

Execution of an algorithm on the MAP requires that the FPGA devices are first configured with the algorithm logic. A first execution of a given subroutine on the MAP performs this configuration. At each invocation of the subroutine, there is a check of the configuration bitstream to be loaded to the MAP. In case, there is no change in the required configuration, the configuration is not repeated. In this case, the time to configure the MAP is amortized over all subsequent calls to the same subroutine.

The execution time measurement on the SRC-6E platform has been performed using three different approaches.

1. Total execution time, including both configuration and data transfer overheads (Total Time). By configuration overhead we mean time necessary to configure system FPGAs. By data transfer overhead we mean time necessary to transfer input and output data between the main microprocessor memory (System Common Memory, SCM) and the MAP's on-board memory (OBM).
2. Total execution time without configuration overhead (Total Time w/o Config).
3. Total execution time for MAP only. This time does not include either configuration or data transfer overheads (MAP Time).

All the SRC-6E time measurements have been done using `second()` routine provided in the MAP compiler library. This routine is based on the number of clock cycles used by the processor.

Table 1 shows the execution time and throughput for all three measurement approaches explained above. A number of encrypted data blocks have been varied from 1024 to 500,000. Each data block is 64-bit (8-byte) long. Column 2 shows the total execution time including both configuration and communication overhead. The corresponding throughput is calculated as a ratio of the number of encrypted data blocks (in Mbytes) to the total execution time in seconds. The results are given in column 3.

Column 4 shows the total execution time without configuration overhead. By subtracting column 4 from column 2, we can find the configuration time for the FPGA on the MAP board. This time is approximately equal to 100 milliseconds. As we can see from column 5, there is a significant increase in the system throughput when we avoid configuration time.

In column 6, the execution time for MAP only is provided. This time does not include any configuration or communication overheads. Column 7 shows a very large increase in the throughput of the system. The MAP implementation of the Triple DES algorithm is pipelined and therefore creates an output block every clock cycle. This was demonstrated by the MAP throughput of 799.8 MB/s. Nevertheless, since the configuration and data transfer overheads were not considered, this measurement can only show the data processing throughput for the FPGA itself, and not for the entire system.

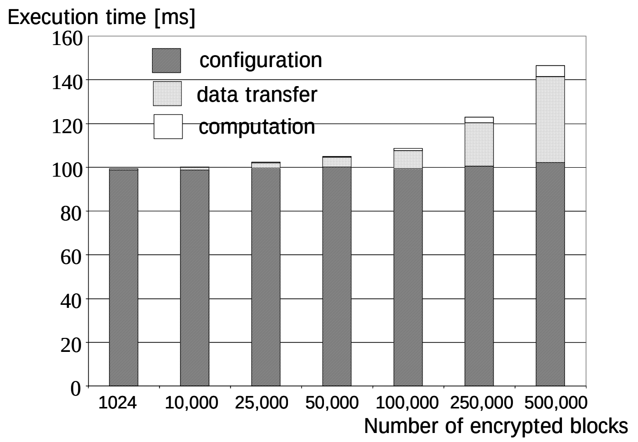

Figure 5. Components of the total execution time as a function of the number of encrypted blocks for high-throughput Triple DES encryption

Table 1. Execution time and Throughput for three different measurement approaches

Length (words)	Total time (sec)	Throughput (MB/sec)	Total time w/o config (sec)	Throughput w/o config (MB/sec)	MAP time (sec)	MAP Throughput (MB/sec)
1024	0.099	0.08	0.00050	16.29	1.12E-05	730.12
10,000	0.100	0.80	0.00133	60.33	0.000101	792.23
25,000	0.102	1.96	0.00266	75.19	0.000251	796.88
50,000	0.105	3.81	0.00492	81.30	0.000501	798.44
100,000	0.108	7.37	0.00932	85.84	0.001001	799.22
250,000	0.123	16.27	0.02228	89.77	0.002501	799.69
500,000	0.146	27.32	0.04421	90.48	0.005001	799.84

In Figure 5, components of the total execution time for Triple DES encryption are shown as a function of the number of processed data blocks. It can be seen that for all considered numbers of blocks, configuration time dominates the entire execution time. However, even if the configuration is done in advance or is amortized over encryption of multiple messages, FPGA devices are still relatively poorly utilized. This is because more time is spent on transferring data between the microprocessor board and the MAP board than on the FPGA computations themselves.

4.2. Intel Pentium 4 measurements

We have run a public domain code of Triple DES on a personal computer equipped with one 1.8 GHz Pentium P4 processor with 512KB cache and 1GByte memory. Two cases have been considered: C implementation of the algorithm and an optimized assembly language implementation of the algorithm.

4.2.1. C Code for Triple DES (Non-optimized): In the non-optimized case, C code is compiled with the Intel C++ compiler v. 5.0 with the -O3 level optimization. The results are given in the left part of Table 2. In contrast to the SRC-6E platform, there is no significant dependence between the throughput and the number of input blocks. This is because all blocks are processed sequentially, one at a time.

Table 2. Total execution time of Triple DES for Pentium 4 processor using optimized and non-optimized DES code

	P4 non-optimized		P4 optimized	
Length (words)	Total time (sec)	Throu-ghpu (MB/sec)	Total time (sec)	Throu-ghput (MB/sec)
1024	0.00379	2.15920	0.00102	8.06299
10,000	0.03663	2.18400	0.01010	7.92354
25,000	0.09279	2.15540	0.02561	7.80969
50,000	0.18637	2.14627	0.05116	7.81937
100,000	0.37150	2.15343	0.09960	8.03253
250,000	0.91990	2.17415	0.25478	7.84985
500,000	1.83200	2.18341	0.49841	8.02546

4.2.2. Assembly Code for Triple DES (Optimized): An optimized implementation of Triple DES considered in this paper is based on [5]. It contains a mixture of the C code and assembly language code. The entire program is compiled using GNU “gcc” version 2.96 20000731 (Red Hat Linux 7.3 2.96-112) with the -O4 optimization option. The results are given in the right part of Table 2. As we can see from Table 2, the total execution time on Pentium P4 decreased by a factor of approximately four as a result of moving majority of computations from C to assembly language.

5: Comparisons for the Triple DES Application

Based on the measurements described in Section IV, the speed-ups of the SRC-6E machine vs. Intel Pentium 4 PC are given in Table 3. Two cases are considered for the Pentium 4 implementation of Triple DES, non-optimized implementation described in Section 4.2.1 and optimized implementation described in section 4.2.2. In both cases, the speed-up increases as a function of the number of data blocks processed, and is the highest for the largest considered input of 500,000 data blocks.

For the case of optimized Pentium assembly language implementation, when all overheads of the SRC-6E machine are included, the SRC-6E platform is approximately 3.5 times faster than Pentium 4. Without configuration time, the speed-up exceeds 11. Without configuration or communication overheads (MAP only), the speed-up of SRC-6E reaches 100. For the case of non-optimized Pentium C implementation, all SRC speed-ups are approximately four times larger.

In Figure 6, the throughput curves for both SRC-6E MAP and the Intel Pentium processor are given. For the reconfigurable computer, the throughput rates are given for two cases. In the first case, all overheads are taken into account. In the second case, the configuration time is

Table 3. Speed-ups of SRC-6E vs. Pentium 4 for high-throughput Triple DES encryption

Length (words)	MAP vs. Non-optimized P4			MAP vs. Optimized P4		
	Speedup Total	Speedup Total w/o Config	Speedup MAP	Speedup Total	Speedup Total w/o Config	Speedup MAP
1024	0.04	7.5	338.2	0.01	2.0	90.6
10000	0.37	27.6	362.8	0.10	7.6	100.0
25000	0.91	34.9	369.7	0.25	9.6	102.0
50000	1.78	37.9	372.0	0.49	10.4	102.1
100000	3.42	39.9	371.1	0.92	10.7	99.5
250000	7.49	41.3	367.8	2.07	11.4	101.9
500000	12.51	41.4	366.3	3.40	11.3	99.7

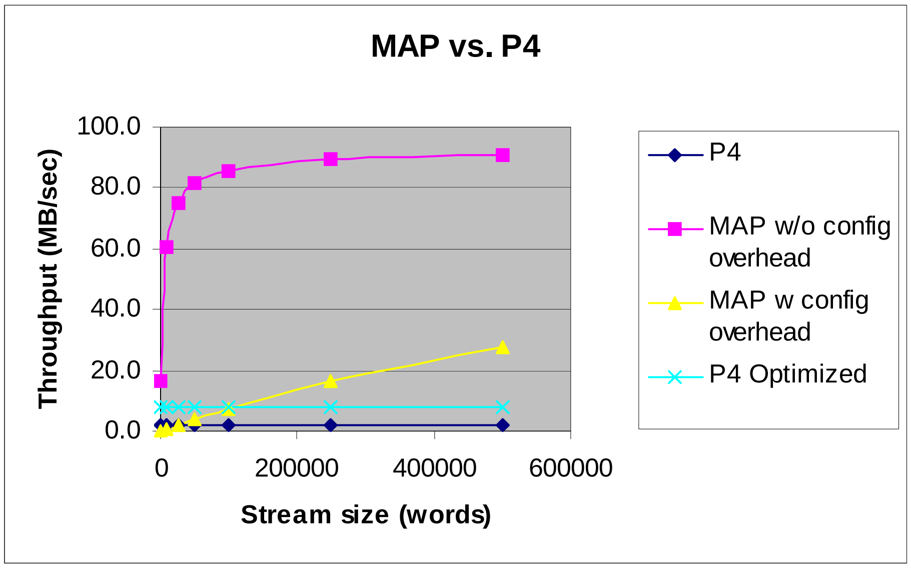

Figure 6. Throughput curves for both SRC-6E MAP and Pentium 4 processor

excluded. Additionally, two throughputs of the Pentium 4 processor are given; one for the optimized program and the second for the non-optimized C only P4 program.

6: Execution time measurements for DES breaker application

As a second benchmark for SRC-6E MAP vs. P4, we have employed the DES Breaker based on the exhaustive key search algorithm. In this algorithm, the entire key space is searched for a match between a given pair of a ciphertext block and the corresponding plaintext block. In each iteration of the algorithm, a given plaintext block is encrypted using a candidate key, and the result is compared against the expected ciphertext. If the match is not found, the key is incremented by one, otherwise the algorithm returns the key as a final solution.

We have measured the performance of the DES Breaker application for both the Pentium 4 and SRC-6E platforms. For the SRC-6E platform, we have made the time measurements based on three different approaches that are defined in previous sections. The difference is that in this application we take scalability and flexibility advantages of reconfigurable computers into account by using more than one DES unit for the key search in FPGAs. These measurements are given in Table 4 where nX refers to a number of DES units operating in parallel within MAP.

In Figure 7, the components of the total execution time are presented for the case of a single DES unit. As we can see, a vast majority of time is spent for actual computations, a small fixed amount of time for configuration, and almost no time for data transfer. This is because all new inputs (new keys) are computed on the MAP board itself and do not need to be transmitted from the microprocessor board. This is the most favorable scenario from the point of view of performance of SRC-6E.

As a second step, we have implemented DES Breaker application in two different ways on Pentium 4 1.8GHz PC, using the same hardware and software environments as in the case of the Triple DES benchmark. As one way of implementing DES Breaker, we have used the non-optimized implementation of DES, coded entirely in C, described in Section 4.2.1. In this implementation, both pre-calculation of the round keys and DES encryption are coded in C. The results are given in Table 5. As a second way of implementing DES Breaker application, we have used an optimized version of the DES P4 implementation based on [5]. In this case, an optimized assembly language code was used for DES encryption. The results are given in Table 5. Unfortunately, as we discovered, the optimized version of DES has been optimized specifically for encryption of long streams of data with the same key, and appeared to be extremely inefficient when the keys needed to be changed for every new input data block. As a result, since the pre-calculation of the round keys takes significantly longer than DES encryption itself, the total execution time of the DES breaker application is longer for the optimized version of the DES code than for the non-optimized version of the DES code.

7: Comparisons for DES breaker application

Using the total execution time measurements, the results of which are given in Tables 4 and 5 (second column), we easily derived speedup factors for SRC-6E vs. a PC based on Pentium 4 processor. Only the case of the non-optimized implementation of DES on Pentium 4 was considered, since the optimized implementation of DES appeared to be significantly less efficient for this application.

Table 4. Execution time and Throughput for three different measurement approaches where nX refers to the number of parallel DES engines in MAP

Number of DES units	Search Size (keys)	Total Time (sec)	Total Time w/o Config. (sec)	MAP only (sec)
1 X	128,000	0.101	0.0016	0.00128
	1,000,000	0.109	0.0103	0.01001
	100,000,000	1.101	1.0006	1.00001
2 X	128,000	0.101	0.0009	0.00064
	1,000,000	0.104	0.0053	0.00500
	100,000,000	0.602	0.5006	0.50000
4 X	128,000	0.101	0.0006	0.00032
	1,000,000	0.102	0.0028	0.00250
	100,000,000	0.352	0.2503	0.25000
8 X	128,000	0.097	0.0005	0.00016
	1,000,000	0.098	0.0015	0.00125
	100,000,000	0.222	0.1253	0.12500

Table 5. Total execution time of the DES Breaker for Pentium 4 processor using optimized and non-optimized DES code

Search size (keys)	Time for non-optimized DES (sec)	Time for optimized DES (sec)
128,000	0.25	3.22
1,000,000	1.97	24.64
100,000,000	198.40	2394.51

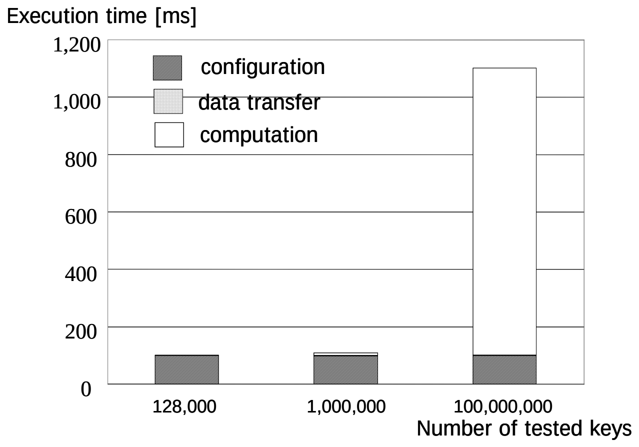

Figure 7. Components of the total execution time as a function of the number of tested keys for the DES breaker application

Table 6. Speedup for SRC-6E vs. Pentium 4 1.8GHz processor for the compiler non-optimized case. Where nX refers to number of parallel DES engines in MAP.

Number of DES units	Search Size (keys)	Speedup Total	Speedup w/o Config.	Speedup MAP only
1 X	128,000	2.5	157.4	194.0
	1,000,000	18.1	191.3	197.3
	100,000,000	180.2	198.3	198.4
2 X	128,000	2.5	265.1	387.8
	1,000,000	18.9	373.0	394.6
	100,000,000	329.4	396.3	396.8
4 X	128,000	2.5	406.7	774.6
	1,000,000	19.3	706.0	789.0
	100,000,000	563.0	792.6	793.6

8 X	128,000	2.6	500.0	1562.5
	1,000,000	20.1	1313.3	1576.0
	100,000,000	893.7	1583.4	1587.2

When 8 units of DES are implemented in parallel on a single FPGA, then even if we take into account all configuration and data transfer overheads, SRC-6E platform is still approximately 894 times faster compared to the C program running on a PC with 1.8 GHz Pentium 4. This speed up factor reaches 1583 when we omit the configuration time. When we consider only MAP (FPGA processor board) the speed up is about 1587.

8: Conclusions

The two benchmarks, Triple DES and DES Breaker represent two distinct classes of algorithms. Both are compute intensive, but they differ in their data transfer characteristics: Triple DES encryption is based on real-time data streaming, while DES Breaker has minimal input/output requirements. In both cases the SRC-6E system outperforms the P4 microprocessor. However, the speed-up factor varies significantly depending on the application type.

For Triple DES as a benchmark, we have demonstrated the overall speed-up of 3 between the SRC-6E machine and the standard PC. When the configuration time of FPGAs in the SRC-6E machine was eliminated the speed-up increased to a factor of 11. When both configuration and communication overheads were eliminated, the speed-up reached a factor of 100.

On the other hand, for the DES Breaker benchmark, an 894x speedup has been achieved even with configuration of the FPGAs present. Eliminating the configuration time yielded a 1583x speedup for SRC-6E over P4. The computational intensity and the relative minimal data movement put the reconfigurable processor at its best advantage.

Based on these results, we clearly see the importance of an overhead management, in particularly eliminating the configuration time from the main computational path. Obviously, the configuration time is not unique to the SRC-6E. It exists for all systems that use FPGAs. Configuration times would be worse for systems that use the serial port for configuration. For the applications that require long execution time, such as DES Breaker, the configuration time overhead can be negligible. Nevertheless, for short and sequential applications, configuration time is a major source of the performance degradation and must be minimized or eliminated. To deliver the performance potential of reconfigurable computing in the general purpose computing arena, the compiler and run time libraries must eliminate configuration time from the computational path. Latency hiding techniques such as preloading configurations during initialization, and ping-pong allocation of reconfigurable chips and processors can be used.

In case of run-time reconfiguration, the run-time switching from one algorithm to another would cause additional demands upon reconfiguration time. The increase in the algorithm switching frequency would make the SRC-6E system inefficient from the execution point of view compared to the standard microprocessors if latency hiding is not utilized.

SRC understands the impact of the reconfiguration of the FPGAs in the MAP and is working on methods that will reduce the apparent configuration time for multiple algorithms using the MAP. Some of the techniques that can be used are flip-flopping the FPGA used by an algorithm. This would mean that two algorithms utilizing single FPGAs can be loaded into a MAP and thereby incurring the configuration only once. There are cases that will require more elegant solutions if there are more than two algorithms using the MAP or for algorithms requiring both FPGAs in the MAP.

There are significant application performance gains to be achieved using run time reconfigurable systems like the SRC-6E. Taking a system wide approach that addresses the programming model, the resource management, and the overhead management will permit these performance gains to be achieved in a wide range of applications.

Acknowledgement

We would like to thank the SRC Technical Support Group for their excellent 24-hour support. Also, we would like to thank the members of High Performance Computing Laboratory at George Washington University for their co-operation.

References

[1] SRC-6E Fortran Programming Environment Guide, SRC Computers, Inc. 2002
[2] William Stallings, Cryptography and Network Security, Prentice Hall, 1999
[3] Tuchman, W. “Hellman Presents No Shortcut Solutions to DES.” IEEE Spectrum, July 1979
[4] Macro Integrator’ s Manual v1.0, SRC Computers, Inc. 2002
[5] Phil Karn, Software Packages and Utilities, http://www.ka9q.net/code/des/index.html.