Opportunities for parallelism when implementing algorithms in VHDL - A Case Study - Shift-Or

Barry Schulz, Chirag Parikh
School of Engineering
Grand Valley State University
Grand Rapids, Michigan 49504
Email: parikhec@gvsu.edu

Christian Trefftz
School of Computing
Grand Valley State University
Allendale, Michigan 49401
Email: trefftzc@gvsu.edu

Abstract—Field Programmable Gate Arrays (FPGAs) are becoming pervasive in High Performance Computing systems, making it possible for developers to implement algorithms (or at least portions of algorithms) directly in hardware using Hardware Description Languages. The Shift-Or algorithm for exact string matching was implemented using VHDL on a XILINX FPGA. This paper discusses some of the opportunities for performance improvement offered by VHDL that were used when implementing the Shift-Or algorithm.

I. INTRODUCTION

Field Programmable Gate Arrays (FPGAs) [1] are becoming pervasive in High Performance Computing systems, making it possible for developers to implement algorithms (or at least portions of algorithms) directly in hardware using Hardware Description Languages. Recent High Performance Computers from SGI and Cray include FPGAs. Software developers for these systems have the opportunity to implement part of their code on the available FPGA.

Algorithms for exact string matching [2] have numerous applications in different fields. Bio-informatics and computer virus detection are just two examples of applications where exact string matching is frequently carried out. There are numerous exact string matching algorithms that have similar performance characteristics. Which algorithm is best depends on the length of the pattern being searched for, the number of letters in the alphabet and the particular architecture where the program is being executed.

Navarro and Raffinot [3] have studied numerous exact string searching algorithms and published a set of guidelines discussing algorithm selection. They identified the Shift-Or [4] as one of the four most competitive algorithms.

This paper describes some lessons learned regarding the parallelism that can be obtained in VHDL/FPGA implementation of a sequential algorithm. A careful examination of the algorithm can lead to a redesign of the way in which the core objectives of the algorithm are achieved. Work is still going on in improving the performance of the implementation, but preliminary results are encouraging and show significant speed improvements over a traditional implementation of the algorithm using C on a PowerPC. Raw performance might not be the only deciding factor, as power consumption becomes a relevant issue as well.

II. THE SHIFT-OR ALGORITHM AND THE PLATFORM USED FOR THE EXPERIMENTS

A. The Shift-Or algorithm

The following notation is used:

- m denotes the length of the pattern x being searched,
- n denotes the length of the string y over which the search is being performed and
- ASIZE denotes the size of the alphabet.

The Shift-Or [4] algorithm uses bit-wise techniques. It is very competitive if the length of the pattern (m) is smaller than the word size of the computer where the algorithm is implemented.

The algorithm includes a pre-processing stage. In the pre-processing stage, a mask is created for every character in the alphabet. In the mask corresponding to character 'c', the bits corresponding to the positions where the character 'c' appears in the pattern are set. The C code for the pre-processing stage follows:
int preShiftOr(char *x, int m, unsigned int S[]) {
    /* x is the pattern */
    /* m is the length of the pattern x */
    /* S is the array of masks */
    unsigned int j, lim;
    int i;
    /* ASIZE is the alphabet size */
    for (i = 0; i < ASIZE; i++)
        S[i] = ~0;
    for (lim = i = 0, j = 1; i < m; i++, j = j << 1)
    {
        S[x[i]] &= ~j;
        lim |= j;
    }
    lim = ~(lim >> 1);
    return(lim);
}

void ShiftOr(char *x, int m, char *y, int n) {
    /* x is the pattern */
    /* m is the length of the pattern x */
    /* y is the string over which the search is conducted */
    /* n is the length of the string y */
    unsigned int lim, state;
    /* ASIZE is the alphabet size */
    /* S is an array of masks, */
    /* one for each character in the alphabet */
    unsigned int S[ASIZE];
    int j;
    /* WORD is the length of an integer */
    /* variable in the platform */
    if (m > WORD)
        error("ShiftOr: Use pattern size <= word size");
    /* Preprocessing */
    lim = preShiftOr(x, m, S);
    /* Searching */
    for (state = ~0, j = 0; j < n; j++)
    {
        state = (state << 1) | S[y[j]];
        if (state < lim)
            OUTPUT(j - m + 1);
    }
}

III. PRE-PROCESSING NOT NECESSARY IN VHDL

When implementing an algorithm using a different computational model, sometimes it can be useful to take a step back and take a global view at the core of the algorithm. The core of the Shift-or algorithm is to track the matches of the characters in the pattern and the string being searched. The masks that are calculated in the pre-processing stage for each one of the characters in the alphabet are really accessory. The experiments conducted by Raffinot and Navarro [3] showed that Shift-or is competitive against other exact string

In the algorithm itself, the variable state that keeps track of the characters that have been matched is initialized to all ones. The string being searched is read from left to right, starting at the leftmost position and continuing until the very last position of the string. When a character is read, its value is used to retrieve its corresponding mask. The state variable is shifted one position to the left. An OR operation is performed between the mask and the variable state. The value of the most significant bit in state becomes 0 when the pattern is found on the string. This condition is detected in the code below with the comparison between the variable state and the variable lim that was returned by the preprocessing stage.

The interested reader is referred to Lecroq’s more detailed description accompanied by an animation (applet) of the algorithm, available at http://www-igm.univ-mlv.fr/~lecroq/string/node6.html

B. The platform used for the experiments

This project was carried out using a XILINX ML310 board. This board comes with an XC2VP30 (Virtex II Pro) FPGA and 256 Megabytes of RAM.

The software used to configure the FPGA was XILINX EDK version 10.1.03 that works in conjunction with XILINX ISE version 10.1.03.

The design for the FPGA was written using VHDL (VHSIC hardware description language).

Currently, in the design, the processor and the VHDL implementation of the function are working at 100 MHz. The processor could go up to 300 MHz, but not the VHDL peripheral. Newer boards include FPGAs that run at faster speeds.

The C code for the Shift-Or algorithm follows:

```c

In the algorithm itself, the variable state that keeps track of the characters that have been matched is initialized to all ones. The string being searched is read from left to right, starting at the leftmost position and continuing until the very last position of the string. When a character is read, its value is used to retrieve its corresponding mask. The state variable is shifted one position to the left. An OR operation is performed between the mask and the variable state. The value of the most significant bit in state becomes 0 when the pattern is found on the string. This condition is detected in the code below with the comparison between the variable state and the variable lim that was returned by the preprocessing stage.

```
searching algorithms for short patterns and small alphabets. The pre-processing stage for a (relatively) large alphabet takes a large portion of the total execution time and makes other algorithms more efficient.

With VHDL it is possible to:

- Calculate the masks for all the characters in the alphabet in parallel or
- Calculate the mask for a character \( c \) that is being read from the string right before it is processed. All the \( m \) characters in the pattern can be compared, in parallel, against the character \( c \) to set the \( m \) bits in the mask to either 0 or 1, depending on whether or not the character \( c \) appears in the corresponding position.
- Calculate the masks for all the characters in the alphabet in parallel and calculating the bits of every mask in parallel (nested parallelism).

With any of the three approaches, the time complexity of the pre-processing stage is significantly reduced. In this project, the second option was chosen, and the mask for the character being read from the input string, is calculated as each character is read.

A small subset of the VHDL code is included here to illustrate the point

```vhdl
-- REPRESENTS THE STORING OF Si
-- WHEN CHECKING THE PATTERN
-- AGAINST EACH TEXT BYTE
if yString (0 to 7) = xWord (0 to 7) then
    Pattern16(15) <= '1';
else
    Pattern16(15) <= '0';
end if;

if yString (24 to 31) = xWord (0 to 7) then
    Pattern13(15) <= '1';
else
    Pattern13(15) <= '0';
end if;
```

Notice that each one of the masks, as well as the variable state, are integers. This implies that the size of an integer on a particular platform will limit the maximum value of \( m \) that can be handled well: On 32 bit microprocessors, integers are usually 32 bits wide, on more recent 64 bit microprocessors, one can choose to use 64 bit integers (long integers). Hence, on 32-bit machines, the maximum value of \( m \) that can be handled efficiently is 32 and on 64-bit machines the maximum value for \( m \) is 64. Leidig [5] compared the performance of the most competitive exact string matching algorithms on 32-bit and 64-bit microprocessors. Some improvements were observed in the speed of the algorithm when using 64 bit microprocessors instead of 32 bit microprocessors and longer patterns could be searched for with 64 bit microprocessors. Notice that these integers are being treated as arrays of bits. Or and shifts are being performed on these arrays of bits, but none of the operations that are regularly used on integers are performed on the masks or the state variable.

On FPGAs, the designer has the freedom to define array of bits of arbitrary length. Therefore, the shift-or implementation could work well with much longer patterns. The native size of an integer becomes irrelevant and it does not restrict the maximum value of the size of the pattern \( m \). The VHDL code that defines the bit of vectors that contains the pattern being searched is:

```vhdl
signal xWord : STD_LOGIC_VECTOR (0 to 127)
```

In this project, the maximum length of the pattern is 16 characters (128 bits / 8 bits per character = 16 characters), but notice that the above segment of VHDL code indicates that this is a constant that could be changed to make it possible to look for patterns that are longer than 16 characters.

V. WIDER REGISTERS

Another constrain that is imposed by the implementation on a regular computing platform has to do with the maximum length of the pattern, \( m \), that can be handled efficiently.

On a FPGA, on the other hand, the designer may choose to compress all of the above operations into a singly clock cycle. Furthermore, it is possible to operate, in a single clock cycle on several characters from the string \( y \). This is equivalent to working on several consecutive values of \( j \) at the same time.

The maximum clock cycle at which the FPGA can operate will impose restrictions on the maximum number of operations that can be performed in a single clock cycle. The bus that
is being used to transfer the characters from memory to the module in the FPGA will pose an additional limitation to the maximum number of characters that can be processed at once. This was the approach taken in this project as well. In every clock cycle, four characters are processed.

Other parts of the system may become the bottleneck for performance. In this particular design, it is possible to transfer only four characters at a time from main memory to the unit that executes the Shift-or function.

VI. CONCLUSIONS AND FUTURE WORK

When implementing an existing algorithm on a FPGA using VHDL, numerous opportunities exist to parallelize the activities required by the algorithm. A careful examination of the core activities in the algorithm is required to identify which of those tasks can be carried out in parallel. A collaboration between Computer Engineers who are proficient in VHDL and Computer Scientists familiar with particular algorithms can be very productive in producing better hardware implementations of existing algorithms that traditionally have been implemented in hardware only.

In the Shift-or algorithm implementation discussed here, the preprocessing stage was eliminated and replaced with a dynamic calculation of the mask of the character being read from the string. Four characters from the string are processed in every clock cycle.

Future work includes implementing other exact string matching algorithms: Horspool, BNDM and BOM [3] and other types of string matching algorithms: Multiple-string, regular expression matching and approximate matching.

REFERENCES


