Using SystemC to Model and Simulate a Many-Core Architecture for LU Decomposition

Ana Rita Silva*, Wilson José*, Horácio Neto†, Mário Véstias‡
*INESC-ID, †INESC-ID/IST/UTL, ‡INESC-ID/ISEL/IPL
anaritasilva@inesc-id.pt, wilsonmaltez@inesc-id.pt, hcn@inesc-id.pt, mvestias@deetc.isel.pt

Abstract

Designing efficient many-core architectures with hundreds of cores is a very challenging task due to the complexity and size of the design space. System-level simulation can help designers exploring the design space of many-core architectures. In this paper, we use SystemC to model a many-core architecture and run a parallel LU decomposition algorithm, an important linear algebra kernel that is widely used in both scientific and engineering applications. The design is parameterized permitting to adapt the model to various hardware constraints. The simulation of the model outputs the number of communications and the number of clock cycles required to complete the algorithm. Given the obtained simulation times, SystemC can be used to efficiently model many-core architectures. Also, the results show the scalability of the architecture and are according to the theoretical formulations.

1. Introduction

Matrix factorization is a key computational kernel in linear algebra. The Lower/Upper (LU) decomposition algorithm is the most used matrix factorization method. It is widely used in scientific and engineering applications to solve dense linear equations and is included in many popular linear algebra libraries such LAPACK [1].

LU decomposition can be implemented on FPGAs to accelerate algorithms in scientific computing, where latency performance of the hardware accelerator is important.

In this work, we map a parallel LU decomposition algorithm on a system with \( p \) processing elements based on the column-oriented LU decomposition [2] and describe the implementation of the system model using SystemC. The design is parameterized so that it can easily adapt to the hardware constraints; the parameters include the number of cores and the local storage size.

Using SystemC, we can model systems above RTL, including systems that might be implemented in software, hardware, or some combination of the two [3].

The higher level of abstraction gives the design team a fundamental understanding, early in the design process, of the behaviour of the entire system and enables better system trade-offs, better and earlier verification, and overall productivity gains through reuse of early system models as executable specifications [4].

The rest of the paper is organized as follows. The next section describes previous proposals of LU decompositions architectures and Section 3 does an overview of LU decomposition. Section 4 describes the many-core architecture to be modeled, the LU decomposition algorithm to be simulated and a theoretical model for the number of communication cycles and operations. Section 5 describes the SystemC model of the architecture. Experimental results are discussed in Section 6. Finally, section 7 concludes the paper.

2. Related Work

There have been some works on FPGA-based matrix factorization. Wang et al. [5] proposed a multiprocessor system on an FPGA device, and used it for LU decomposition of sparse block-diagonal bordered matrices. Each processor on the FPGA is attached to a floating-point adder/subtractor, a floating-point multiplier, and a floating-point divider.

Choi et al. [6] proposed a linear array architecture for fixed-point LU. The array consists of \( n \) processing elements (PEs) and each PE contains a multiplier, an adder/subtractor and a reciprocator.

In [7], a circular array architecture was proposed for floating-point LU decomposition. It uses \( n \) PEs and only the first PE, \( PE_0 \), contains a divider. \( PE_j \) (\( 1 \leq j \leq n-1 \)) contains one MAC (Multiplier and ACumulator) and a storage of size \( n - j \). However, due to the design complexity of the floating-point units, an FPGA device cannot contain \( n \) PEs when \( n \) is larger than several tens.

The authors in [8] give a detailed description of block LU decomposition algorithm and proposes an architecture to run it. The matrix is partitioned into \( b \times b \) blocks, where \( b \) is determined by the number of configurable slices.

3. LU Decomposition

In LU decomposition a \( n \times n \) matrix, \( A \) (\( a_{xy} \)), is decomposed into a lower triangular matrix \( L \) (\( l_{xy} \)) with 0s above the diagonal and 1s on the diagonal, and an upper triangular matrix \( U \) (\( u_{xy} \)) with 0s below the diagonal, both of size \( n \times n \), where \( x \) is the row index and \( y \) is the column index (see figure 1).

Lower/Upper triangular decomposition is performed by a sequence of Gaussian eliminations to form \( A = LU \). In
this work, we assume that matrix $A$ is a non-singular matrix and no pivoting is needed.

In [3], a sequential algorithm for LU decomposition is discussed. It consists of three main steps:

- Step 1: The column vector $a_{s1}$ ($2 \leq s \leq n$) is multiplied by the reciprocal of $a_{11}$. The resulting column vector is $l_{11}$;
- Step 2: $a_{ij}$ is multiplied by the column vector $l_{1i}$. The product is subtracted from the submatrix $a_{ij}$ ($2 \leq i, j \leq n$);
- Step 3: Steps 1 and 2 are recursively applied to the new submatrix generated in Step 2. During the kth iteration, $l_{ik}$ and $u_{ij}$ ($k + 1 \leq i, j \leq n$) are generated. When $k = n - 1$, the decomposition is complete.

A pseudo code for the decomposition is given in Listing 1.

Listing 1. Column Oriented LU Decomposition

```plaintext
for k = 1 to n - 1
  for s = k + 1 to n
    $l_{sk} = \frac{a_{sk}}{a_{kk}}$
  for i = k + 1 to n
    $a_{ij} = a_{ij} - l_{ik} \times a_{kj}$
```

4. Many-Core Architecture and Algorithm Analysis

In this section, we present the algorithm proposed to parallelize the LU Decomposition problem for a system with $p$ cores organized as a 2-dimensional array.

The parallel architecture is organized as a 2D mesh of processing elements (cores). Each core consists mainly of a floating-point divider, a floating-point multiplier and adder/subtractor and a dual-port memory. Access to the external memory is controlled with a direct memory access (DMA) module that can deliver burst transactions with one transfer per cycle. There is only a single interface to external memory, so writing to and reading from external cannot be done in parallel.

To facilitate the presentation of the algorithm, we consider a LU Decomposition of a matrix $A$ with size $6 \times 6$ and a model with 2 processing elements.

Processing element P1 is responsible for calculating the first submatrix generated, $A^r$ (k=1). P1 receives the first column of matrix $A$, $B_0(a_{11}, a_{21}, a_{31}, a_{41}, a_{51}, a_{61})$, and determines the values of the first column of matrix L ($l_{21}, l_{31}, l_{41}, l_{51}, l_{61}$), according to the algorithm in Listing 1.

The operations are initiated at the moment the first value is available and the values $l$ are stored in a local memory and sent to the external memory.

Next, P1 receives the second column, $B_1(a_{12}, a_{22}, a_{32}, a_{42}, a_{52}, a_{62})$, and calculates the new column $B_1'$ ($a_{22}', a_{32}', a_{42}', a_{52}', a_{62}'$) and so on, until we obtain the submatrix $A^r$. The first value of each column (a’22, a’23, a’24, a’25, a’26) is stored in the external memory.

As processing element P1 determines the columns of submatrix $A^r(B_1', B_2', B_3', B_4', B_5')$, the results are sent to the processing element P2.

Processing element P2 is responsible for determining submatrix $A^r(k = 2)$. Similarly to processing element P1, P2 receives the column $B_1'$, obtained by P1, and determines the second column of matrix L ($l_{32}, l_{42}, l_{52}, l_{62}$), stores the values in a local memory and sends it to the external memory, followed by the computation of $B_2''(a_{33}', a_{43}', a_{53}', a_{63}')$ from $B_2'$ and so forth.

Being P2 the last processing element in this example, it sends all the results to the external memory. Then, processing element P1 receives the values of matrix $A^r$, obtained by processing element P2 and determines matrix $A^r''$ and so forth, according to figure 2.

As mentioned above, each processing element stores the values of the matrix L in a local memory to be used in the
calculations of new columns. Thus, each processing element $P_x$ needs a memory location with a size equal to a column-$x$.

The total number of communications is given by equation 1, where the size of matrix $A$ is $n \times n$ and $p$ the number of processing elements.

$$\sum_{k=0}^{n-1} (n-kp)^2 + \sum_{k=1}^{n-1} \sum_{i=0}^{p-1} (n-pi+k) + \sum_{k=1}^{n-1} (n-kp)^2$$

The first parameter of the sum is the number of data sent from memory to the first processing element, $P_1$. The other parameters correspond to the sending of data from processing elements to memory: the second corresponds to the values of matrix $L$, the third to the data matrix $B$ calculated by all processing elements except for the last one and the fourth parameter correspond to the values of matrix $B$ calculated by the last processing element.

The total number of operations is given by equation 2.

$$\sum_{k=1}^{n-1} k + 2 \times \sum_{k=1}^{n-1} k^2$$

The first parameter is the number of divisions needed to calculate the matrix $L$ and the second is the number of multiplications required to compute the matrix $U$, which is equal to the number of subtractions.

## 5. SystemC Model of the Architecture

This section describes the implementation of the system model using SystemC. For ease of explanation, we describe the model considering only four processing elements (see Figure 3)

Figure 3. SystemC model of the many-core architecture with four processing elements.

The system consists of a module MEM representing the DMA and $p$ modules $P_i$ representing the cores. The communication between modules is performed with FIFOs.

The details of communication among modules are separated from the details of the implementation of the functional units.

There is a module Arbiter used to arbitrate competing communication requests and consists of one method process (SC_METHOD). Outstanding requests are passed to the arbiter as a vector of structures. On each cycle, module Arbiter receives all the requests and selects several for execution based on their arbitration policy. Requests are chosen according to the chronological order and attended if FIFOs are free to write.

When a module wants to write to a certain FIFO, it is created and filled a new request using the function fill_req. After writing all data, the request is released using the function unlock_req (see code in Listing 2).

### Listing 2. Functions fill_req and unlock_req

```c
void fill_req(int *i_req, req *v_req, int n, int req_out, int *x){
    v_req[*i_req].n_in=n;
    v_req[*i_req].out=req_out;
    for (int j=0; j<n; j++){
        v_req[*i_req].in[j]=x[j];
    }
    *i_req=*i_req+1;
}

void unlock_req(int n, int *x, int *lock_list){
    for (int j=0; j<n; j++){
        lock_list[x[j]] = 0;
    }
}
```

FIFOs are modeling using the code in Listing 3

### Listing 3. systemC modelling of read and write FIFO functions

```c
void write_fifo(int *f, int *full, double *FIFO, sc_event *ev_in, sc_event *ev_out, double value ,int size){
    if(*full==size) wait(*ev_out);
    FIFO[*f]=value;
    *f=*f+1;
    *full=*full+1;
    ev_in->notify();
    if (*f==size) *f=0;
}

double read_fifo(int *full, int *il, double *FIFO, sc_event *ev_in, sc_event *ev_out, int size){
    double x;
    if(*full==0) wait(*ev_in);
    x=FIFO[*il];
    return x;
}
```
6. Simulation and Results

To implement this model with a given number of cores, we developed a program in C to generate the respective code in SystemC. Using this program, we can implement systems with several cores to simulate the LU decomposition for square matrices of different sizes.

The number of cycles is obtained by considering that each operation has a throughput of one result per cycle and the number of cycles need to write data is one cycle and FIFOs with sufficient size to obtain the fewest number of cycles.

Tables 1 to 3 show the results obtained in regard to the number of communications, the total number of cycles and the simulation time, considering square matrices of different sizes and different number of cores.

Table 1. Number of communications, execution cycles and simulation times of LU decomposition with 16 cores

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>Communications</th>
<th>Exec. Cycles</th>
<th>Simul. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>2.512</td>
<td>2.541</td>
<td>1s</td>
</tr>
<tr>
<td>64</td>
<td>15.200</td>
<td>15.234</td>
<td>7s</td>
</tr>
<tr>
<td>128</td>
<td>103.872</td>
<td>104.017</td>
<td>34s</td>
</tr>
<tr>
<td>256</td>
<td>763.776</td>
<td>765.216</td>
<td>3 m</td>
</tr>
<tr>
<td>512</td>
<td>5.848.832</td>
<td>5.853.972</td>
<td>18 m</td>
</tr>
</tbody>
</table>

Table 2. Number of communications, execution cycles and simulation times of LU decomposition with 32 cores

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>Communications</th>
<th>Exec. Cycles</th>
<th>Simul. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>10.144</td>
<td>10.177</td>
<td>2s</td>
</tr>
<tr>
<td>128</td>
<td>61.120</td>
<td>61.164</td>
<td>35s</td>
</tr>
<tr>
<td>256</td>
<td>416.640</td>
<td>416.824</td>
<td>2 m</td>
</tr>
<tr>
<td>512</td>
<td>3.059.456</td>
<td>3.061.743</td>
<td>18 m</td>
</tr>
</tbody>
</table>

Observing the results, we conclude that the number of communications is consistent with equation 1. Considering the example of matrices with size $32 \times 32$ and 16 processing elements the total number of communications if given by equation 3.

\[
\sum_{k=0}^{n-1} (n-kp)^2 + \sum_{k=0}^{n-1} k + \sum_{k=1}^{n-p-1} (\sum_{i=1}^{n-p-1} (n - pi + k)) + \sum_{i=1}^{n-p-1} (n - kp)^2 = 1280 + 496 + 480 + 256 = 2512
\]

Figures 4 and 5 compares the number of communications and the total number of cycles with the size of matrix $A$, considering 16 and 64 cores, respectively.

According to both figures, we verified that the number of communications and the number of cycles increases exponentially with the increasing size of matrix $A$.

The number of communications is practically equal to the number of cycles, concluding that the number of computations can be totally overlapped with the communications.

Looking at tables 1, 2 and 3, we found that the number of communications and the total number of cycles decreases with the increase of the number of processing elements (see figure 6).

As can be seen, the larger the size of the matrix, the more accentuated is the variation in the number of communications.

Table 4 shows the total number of operations needed to perform the algorithm.

Observing the results, we can conclude that the number of computations is consistent with equation 2. Also, the increase in the number of operations is according to the complexity of the algorithm ($n^3$).
From the results we can determine the performance efficiency (ratio between sustained performance and peak performance) of the architecture. For example, considering a matrix of size $512 \times 512$, we achieve efficiencies up to 46%.

### 7. Conclusions and Future Work

In this paper we have described a many-core architecture using SystemC to perform LU decomposition. We started with a brief overview of LU decomposition, describing in particular the Column-Oriented method. The algorithm to execute on a 2-dimensional multiprocessor array was presented and analyzed theoretically in regard to the number of communications and computations. We simulated the model to evaluate number of communications and total number of clock cycles required for the complete algorithm execution. The results show the scalability of the architecture and are according to the theoretical formulations.

Given the obtained simulation times, SystemC is a good candidate to efficiently model many-core architectures. Therefore, the modeling process with SystemC is being applied to other parallel algorithms.

### Acknowledgment

This work was supported by national funds through FCT, Fundação para a Ciência e Tecnologia, under projects PEst-OE/EEI/LA0021/2013 and PTDC/EEA-ELC/122098/2010.

### References


### Table 4. Number of Computations for different matrix sizes

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th># Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>21,328</td>
</tr>
<tr>
<td>64</td>
<td>172,704</td>
</tr>
<tr>
<td>128</td>
<td>1,389,888</td>
</tr>
<tr>
<td>256</td>
<td>11,152,000</td>
</tr>
<tr>
<td>512</td>
<td>89,347,328</td>
</tr>
</tbody>
</table>