Future Generation Computer Systems 27 (2011) 850–859

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Collaboration of reconfigurable processors in grid computing: Theory and application

Mahmood Ahmadi [a,c,*], Asadollah Shahbahrami [b], Stephan Wong [a]

[a] Computer Engineering Laboratory, Delft University of Technology, 2628 CD, Delft, The Netherlands

[b] Department of Computer Engineering, Faculty of Engineering, P.O. Box: 3756-41635, University of Guilan, Rasht, Iran

[c] Department of Computer Engineering, Faculty of Engineering, University of Razi, P.O. Box: 67149-67346, Kermanshah, Iran

ARTICLE INFO

Article history:
Received 28 May 2010
Received in revised form
7 October 2010
Accepted 25 October 2010
Available online 10 November 2010

Keywords:
Reconfigurable architectures
Grid computing
Multimedia kernels
High-performance computing

ABSTRACT

Traditional grid networks employ General Purpose Processors (GPPs) as their main processing elements. Incorporating reconfigurable processing elements in such networks can be a promising technology to increase their performance. In this paper, we propose and simulate collaboration of reconfigurable processors in grid computing. Collaborative Reconfigurable Grid Computing (CRGC) employs the availability of any reconfigurable processor to accelerate compute-intensive applications such as multimedia kernels. We explore the mapping of some compute-intensive multimedia kernels such as the 2D DWT and the co-occurrence matrix in the CRGC. These multimedia kernels are simulated as an independent set of gridlets submitted to a software simulator called CRGridSim. In addition, we analyze the lower and upper bounds of performance for CRGC. Our experimental results show that the CRGC approach improves performance up to 7.2× and 2.5× compared to a single GPP and the collaboration of GPPs, respectively, when assuming a speedup of 10 of the reconfigurable processors in a grid with 4 nodes.

1. Introduction

In recent years, a significant paradigm for high performance computing has emerged as grid networks. In a (computational) grid network, many heterogenous computing resources are distributed over a large network to efficiently perform computations for one or several large-scale (scientific and/or engineering) applications. The computing resources range from general-purpose processors (GPPs) to application-specific processors or even dedicated hardware circuits depending on the required flexibility or performance, respectively. Grid networks differ from distributed computing networks in the sense that grid networks provide mechanisms within the network to perform resource allocation and scheduling to meet the requirements of the supported application(s). The trade-off between flexibility and performance by using fixed hardware (a GPP is still a fixed processor as its basic functions do not change over time even though its overall functionality can be changed by reprogramming the sequence of basic functions) is the same as witnessed in the field of computer architecture design. [1–4]. Therefore, GPPs are considered highly flexible due to their programmability, but also slow due to the sequential programming nature. Recently, reconfigurable computing has emerged as a solution to "ride" the curve of the mentioned trade-off in a much more gradual manner by utilizing a single flexible and reconfigurable solution. A common approach in reconfigurable computing is to tie a GPP with a reconfigurable co-processor to which highly parallel operations are offloaded [5–7,4]. Such a reconfigurable (co-)processor is usually implemented on a field-programmable gate array (FPGA) fabric that mainly consists of many logic cells connected to each other using an elaborate on-chip interconnection network. Usually utilizing a hardware description language (HDL) already many applications in, e.g., multimedia, bioinformatics, and cryptography, were mapped on FPGAs and their performance improvement demonstrated.

In this paper, we propose the approach of Collaborative Reconfigurable Grid Computing (CRGC) that introduces reconfigurable processors in processing (grid) nodes and a scheme that allows for different nodes to collaborate together in processing a single application. The main concept lies in the fact that the reconfigurable processors adapt themselves to the needed processing requirements (and functionality) without the need to introduce fixed hardware accelerators to improve the overall grid performance. In addition, we analyze the lower and upper bounds of performance for the CRGC. Furthermore, we introduced the neighborhood concept as the collaboration concept between neighboring nodes in order to limit the communication throughout the whole grid. To investigate this concept, we introduced a set of primitives (in which

* Corresponding author at: Computer Engineering Laboratory, Delft University of Technology, 2628 CD, Delft, The Netherlands. Tel.: +31 626971735.
E-mail addresses: m.ahmadi@tudelft.nl (M. Ahmadi), shahbahrami@guilan.ac.ir (A. Shahbahrami), j.s.s.m.wong@tudelft.nl (S. Wong).

0167-739X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2010.10.014

the communication could be simulated) and adapted the grid simulator GridSim v4 to simulate the CRGC concept together with the neighborhood concept. We termed the adapted simulator CRGridSim. To obtain results of real cases, we explored the mapping of two computationally intensive multimedia kernels: the Discrete Wavelet Transform (DWT) and the co-occurrence matrix. These kernels were subdivided into gridlets for simulation in the CRGridSim simulator. Our experimental results show that speedups of up to 7.2 can be achieved when comparing to a grid with only a general-purpose processor and assuming a per-processor speedup of a factor of 10 when comparing executing the same kernel in a GPP or a reconfigurable processor. The main contributions of our work are the following:

- In order to benefit both advantages of grid computing and reconfigurable architectures to achieve high-performance and flexibility, the collaborative reconfigurable processors on grid environments is proposed.
- A performance model analysis for the proposed approach for both lower and upper bounds is investigated.
- A grid simulator is extended for the proposed approach. In addition, several compute-intensive multimedia kernels are studied and mapped using collaboration of four processing elements in grid computing based on neighborhood policy.

This paper is organized as follows. Section 2 presents related work. We describe the collaboration of reconfigurable processors in a grid network in Section 3. In Section 4, we explain the chosen compute-intensive multimedia kernels in more detail. The simulation environment and tools are described in Section 5 followed by a discussion of the evaluation results in Section 6. Finally, conclusions are drawn in Section 7.

2. Related work

In this section, we take a brief look at the related work regarding high-performance reconfigurable computing. In [6], the design and implementation of a metacomputer based on reconfigurable hardware was presented. The Distributed Reconfigurable Metacomputing (DRMC) is defined as "the use of powerful computing resources transparently available to the user via a networked environment". The DRMC provides an environment in which computations can be constructed in a high-level manner and executed on clusters containing reconfigurable hardware. In the DRMC architecture, applications are executed on clusters using the condensed graphs model of computation that allows the parallelism inherent in applications to be executed by representing them as a set of graphs.

In [8], a performance model for the fork-join class and the Synchronous Iterative Algorithm (SIA) was presented. They considered the division of computation between the workstation processor and a reconfigurable processor. They focused on algorithms and applications that fit the fork-join class and SIA types. A set of rules and guidelines for the implementation of Distributed Processing Networks (DPN) as the basis for a dynamic reconfigurable architecture to improve the performance of microprocessor-based systems in computationally intensive application domains was presented in [7]. The DPN presents a general architecture to solve complex problems using reconfigurable computing. It does not include any result to show the possibility of the proposed solution. In [9], the 2D-FFT application was implemented on both a standard cluster and the prototype Adaptable Computing Cluster (ACC). The ACC is an architecture that attempts to improve high-performance cluster computing with FPGAs, but not by merely adding reconfigurable computing resources to each node. Rather, by merging cluster and reconfigurable technologies and enhancing the commodity network interface. In [10], performance of a single reconfigurable processor for grey level co-occurrence matrix (GLCM) and Haralick texture feature for image sizes 512×512, 1024×1024 and 2048×2048 was presented. Speedups of 4.75 and 7.3 were obtained when compared with a GPP. The target hardware for this work was the Celoxica RC1000-PP PCI-based FPGA development board equipped with a Xilinx XCV2000E Virtex FPGA. In addition, a co-occurrence matrix media kernel has been implemented on the various FPGA devices such as Virtex2 and Spartan3 and on a media-enhanced GPP using MMX technology in [11]. Speedups of 20 were obtained using FPGA implementations over media-enhanced GPPs, for an image size 512×512.

In [12], a limited flooding approach for mobile ad-hoc networks was proposed that utilized neighbors information to limit broadcast redundancies. It reduced the control overhead of ad-hoc routing protocols and introduced some benefits including efficient flooding, density, and mobility adaptation. A system infrastructure that allows local mobile devices to interact with the grid is introduced in [13]. A proxy with the ability of dual communication to transfer the request from the mobile device to the grid is the main part of that architecture.

In [14], TeraGrid as an open scientific infrastructure was presented. It consists of eleven partner sites to create an integrated and persistent computational resource. TeraGrid resources encompasses more than a petaflop of computing capability and more than 30 petabytes of online and archival data storage, with rapid access and retrieval over high-performance networks. TraGrid provides limited FPGA resources for its users. These resources consist of an SGI 450 (Brutus) with two RC100 FPGA blades, totaling 4 available FPGAs.

In [15,16], a Distributed ASCI Supercomputer (DAS) as an experimental testbed for research on wide-area distributed and parallel applications was presented. The system was built for the Advanced School for Computing and Imaging (ASCI), a Dutch research school in which several universities participate. The goal of DAS is to provide a common computational infrastructure for researchers within ASCI, who work on various aspects of parallel and distributed systems, including communication substrates, programming environments, and applications. The DAS-3 is a new grid infrastructure for experimental computer science research in The Netherlands. The system will have five clusters with about 270 dual-cpu node supercomputers in total, which are integrated into a single large-scale distributed system using multi-color optical networking capabilities. The DAS-4 is going to prepare and utilize the FPGA as part of its resources [16].

In [17,18], we investigated the concept of collaboration of reconfigurable processors in grid computing. In these papers, we simulate several computationally intensive media kernels and map them on the proposed architecture. The experimental results show that collaboration of reconfigurable processors in grid computing achieves much more performance than the collaboration of GPPs. In the current work, we further investigate the performance of the CRGC by looking at realistic loads and real kernels execution characteristics. In addition, we also determine the lower and upper bounds of performance.

3. Collaboration of reconfigurable processors in grid computing

In this section, we present the concept of collaborative reconfigurable processors and an analysis of lower and upper bounds of performance.

3.1. Collaborative reconfigurable architectures on grid: concept and method

In grid computing, a large pool of heterogeneous computing resources is geographically dispersed over a large network, e.g., the

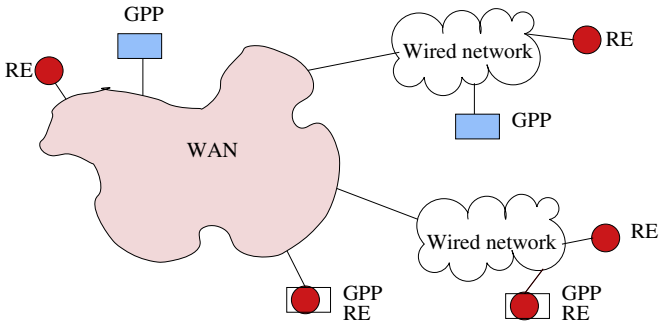

Fig. 1. A general view of a collaborative grid environment with reconfigurable processors and general-purpose processors. The light blue box shows GPP and the box with the circle inside shows GPP augmented with RE (reconfigurable element).

Internet. Our approach to achieve high-performance and flexibility is to utilize reconfigurable processors. It should be noted that the CRGC is different from distributed computing. Distributed computing normally refers to managing or pooling a set of computer system which individually are more limited in their memory and processing power. CRGC (grid computing) is focused on the ability to support computation across multiple administrative domains that sets it apart from traditional distributed computing. Grids offer a way of utilizing the information technology resources. We termed the utilization of reconfigurable processors that collaborate together in a grid environment collaborative reconfigurable grid computing (CRGC). The general view of the CRGC is depicted in Fig. 1. This figure shows that the processing elements are the part of resources in a grid network. Each processing element can be either a GPP or a Reconfigurable Element (RE) also known as a reconfigurable processor. It should be noted that the RE is implemented on an FPGA to execute an amenable application with inherent level of parallelism. The same application can take much longer to execute on a GPP.

Processing elements offload part of their computational workload to reconfigurable computing resources. In this type of computing, various software codes targeting different processing architectures are stored either in a centralized or a decentralized manner and must be distributed to the computing resources when needed. On the other hand, in CRGC, processing elements communicate and collaborate together based on the *neighborhood concept* [18,17]. Each grid processing element requests assistance from neighboring processing elements. The tasks can be inserted into the grid through existing grid elements. In our implementation of the neighborhood concept, the neighbor processing elements are a direct neighbor to a requesting grid element. The direct neighbor is defined as a grid element that is physically (or geographically closely) located next to the current requesting grid element. The network backbone can be seen as a collection of primitives. A primitive is defined as a set of collaborator processing elements, requesting processing elements with related communication links and its network equipment, e.g., routers and switches. Two important primitives are depicted in Fig. 2. The requesting processing elements are also called as main processing elements.

Fig. 2(a) depicts a primitive with one requesting processing element (main processing element) and n collaborating processing elements. In this figure, resource 0 is requesting processing element and resource 1 to resource n are collaborator processing elements. These processing elements and routers are represented as $\{(resource\ 0, router\ 0), (resource\ 1, router\ 1), \ldots, (resource\ n, router\ n)\}$. A primitive with two requesting processing elements and one collaborating processing element is depicted in Fig. 2(b). In this figure, resource 0 and resource 1 are the main processing elements and resource 2 is the collaborator to execute their tasks.

The neighborhood concept with active primitives in a real grid is depicted in Fig. 3.

Based on Fig. 3, we can observe that each user and the related requesting processing element can find the corresponding neighbor processing element. For example, in Fig. 3 user 0 and resource 0 can operate based on the primitives in Fig. 2(a). Therefore, in Fig. 3 in the first scenario, resource 0 is assisted by resource 1 and in the second scenario, resource 0 is assisted by resources 1 and 2. We have a similar condition for users 1 and 2, in this case resource 3 and resource 4 are assisted by resource 5.

3.2. Analysis of lower and upper bounds of performance

This analysis is performed to determine the performance bounds in terms of execution time for the collaborative reconfigurable processors in grid environments. The maximum and minimum execution times show the lower and upper bounds of performance, respectively. Table 1 describes all notations and symbols that were used in the equations.

The total processing time in a non-collaborative system without inter-task dependency can be represented as Eq. (1).

$$t_{non\text{-}col} = \sum_{i=1}^{n} M_i t_{GPP} \tag{1}$$

where $t_{non\text{-}col}$, M_i, and t_{GPP} are the processing time in a non-collaboration system, the number of instructions of subtask i (each

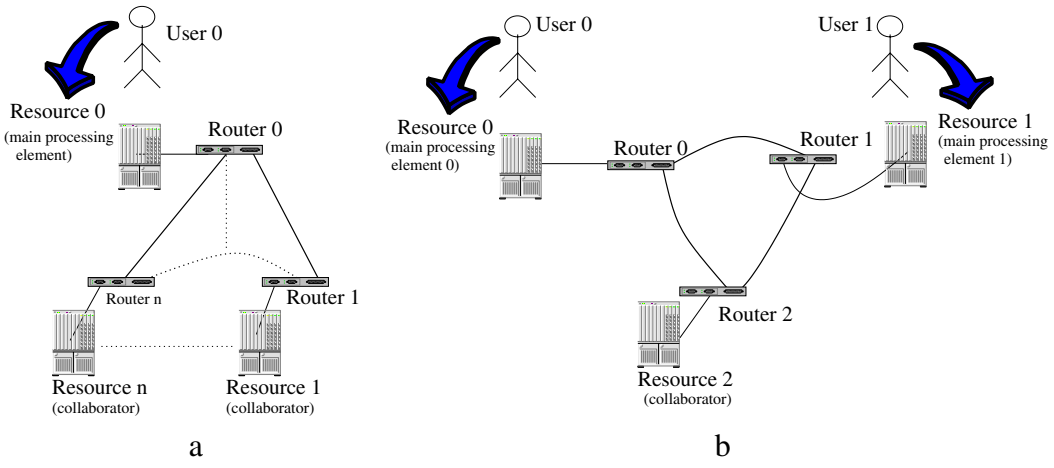

Fig. 2. Basic primitives that are utilized in the neighborhood concept. (a) A primitive with one requesting (main) processing element and n collaborator processing elements. (b) A primitive with two requesting (main) processing elements and one collaborator processing element.

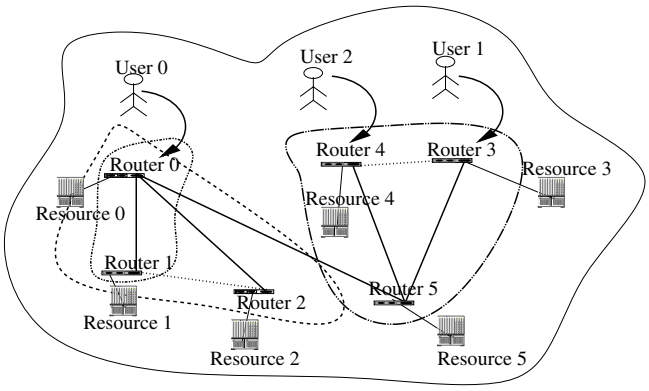

Fig. 3. Active primitives in the sample network.

Table 1
Performance analysis symbols and their definitions.

Symbol	Definition
Bw	Network bandwidth
k	Speedup factor of a reconfigurable processor over a GPP
k_j	Speedup factor of reconfigurable processor j over a GPP
k_{max}	Maximum speedup factor of a reconfigurable processor over a GPP
M_i	Number of instructions for subtask i
n	Number of subtasks (gridlets)
$n1$	Number of subtasks that can be processed by neighbor collaborators
$n1_j$	Number of processed subtasks by collaborator j
n_{pkt}	Number of packets
p	Number of collaborator processing elements
rs	Reconfiguration speed (bit/s)
rfs	Reconfiguration file size (bits)
S	Size of all subtasks (Bytes)
s_i	Size of subtask i (Bytes)
s_{Pkt}	Packet size
t_{col}	Total execution time in a collaboration system
t_{com}	Communication time between different processing elements
t_{GPP}	Processing time for each instruction by a GPP
t_{max}	Maximum execution time of an application on CRGC
t_{min}	Minimum execution time of an application on CRGC
$t_{non\text{-}col}$	Total execution time in a non-collaboration system
t_{PE_j}	Processing time of a subtask using collaborator processing element j
t_{ptt}	Transmission time of a packet
t_{RE}	Processing time for each instruction by a RE that is equal to t_{GPP}/k
t_{rec}	Reconfiguration time

submitted task is broken into several subtasks), and the processing time of each instruction by a GPP, respectively. In addition, the total processing time in a collaborative system is defined in Eq. (2). This is because, in the CRGC system some of the tasks are processed by a GPP and others are distributed among collaborator processing elements. Therefore, the total processing time is the summation of processing time by GPP and the maximum of processing time by the collaborator processing elements.

$$t_{col} = \sum_{i=1}^{n-n1} M_i t_{GPP} + max\{t_{PE_j}, j = 1 \ldots p\} \quad \text{and} \quad n1 = \sum_{j=1}^{p} n1_j \tag{2}$$

where t_{col} is the processing time in a collaborative system, $n1$ is the number of subtasks that are processed by the collaborator processing elements, and $n1_j$ is a set of processed subtasks by an individual collaborator element. Moreover, t_{PE_j} represents the processing time using collaborator processing element j that is defined using the following Eq. (3).

$$t_{PE_j} = \left(\sum_{i=start_j}^{end_j} M_i t_{RE_j} \right) + t_{rec_j} + t_{com_j} = \frac{\sum_{i=start_j}^{end_j} M_i}{k_j} t_{GPP} + \frac{rfs_j}{rs_j} + t_{com_j}. \tag{3}$$

In this equation, the subtasks from $start_j$ to end_j ($end_j - start_j = n1_j$) are processed by collaborator j. The size of the reconfiguration file and the reconfiguration speed are represented by rfs and rs, respectively. In addition, the term t_{rec_j} is the reconfiguration time needed to reconfigure the collaborator processing element j.

The communication time for processing element j is also represented by t_{com_j}. If the submitted tasks cannot be distributed on the collaborator elements then $t_{com} = 0$, $t_{rec} = 0$, $t_{col} = 0$, and $n_1 = 0$ in Eq. (3). As a result, $t_{PE_j} = 0$ and Eq. (2) will be the same as Eq. (1). Therefore, the main processing element (GPP) executes all submitted subtasks without any assistance from neighbor collaborator processing elements. Consequently, the lower bound of performance in terms of maximum execution time (t_{max}) is represented as:

$$t_{max} = \sum_{i=1}^{n} M_i t_{GPP} = t_{non\text{-}col}. \tag{4}$$

To evaluate the upper bound, Eq. (2) should be minimized. In this case, some submitted tasks are executed by reconfigurable processors and the rest are executed on the main processing element. The upper bound of performance in terms of minimum execution time is represented as follows:

$$t_{min} = min \left\{ \sum_{i=1}^{n-n_1} M_i t_{GPP} + max\{t_{PE_j}, j = 1 \ldots p\} \right\}. \tag{5}$$

The minimization of this equation depends on different parameters such as the number of collaborators processing elements, the length and number of submitted subtasks, the reconfiguration time, the bandwidth of the network, the propagation delay, and scheduling algorithm. To simplify the mathematical analysis, we utilize the transitivity property of inequalities to achieve the upper bound of the performance [19].

$$\begin{aligned} t_{min} > min\{t_{PE_j}\} &= min \left\{ \sum_{i=start_j}^{end_j} M_i t_{RE_j} + t_{rec_j} + t_{com_j} \right\} \\ &= min \left\{ \frac{\sum_{i=start_j}^{end_j} M_i}{k_j} t_{GPP} + \frac{rfs_j}{rs_j} + t_{com_j} \right\} \\ &\text{with } end_j - start_j = n1_j \\ &\text{and } j = 1 \ldots p. \end{aligned} \tag{6}$$

We make some reasonable assumptions e.g., t_{rec} and t_{com} are assumed to be very small and these parameters as said before can be ignored. Consequently, Eq. (6) is represented as:

$$t_{min} > min \left\{ \sum_{i=start_j}^{end_j} M_i \frac{t_{GPP}}{k_j}, j = 1 \ldots p \right\}. \tag{7}$$

The right side of Eq. (7) includes $end_j - start_j = n1_j$ subtasks while for p collaborator processing elements with n subtasks this equation is rewritten as follows.

$$min \left\{ \sum_{i=start_j}^{end_j} M_i \frac{t_{GPP}}{k_j}, j = 1 \ldots p \right\} \approx \frac{1}{p} min \left\{ \sum_{i=1}^{n} M_i \frac{t_{GPP}}{k_j}, j = 1 \ldots p \right\}. \tag{8}$$

```
void DWT_Daub_4()
{
  int i, j, jj, ii;
  float low[] ={-0.1294, 0.2241, 0.8365 , 0.4830};
  float high[]={-0.4830, 0.8365, -0.2241, -0.1294};

  for (i=0; i<N; i++)
    for(j=0, jj=0; jj<M; j++, jj +=2)
   {
      tmp[i][j]  = img[i][jj]     * low[0] +
                   img[i][jj + 1] * low[1] +
                   img[i][jj + 2] * low[2] +
                   img[i][jj + 3] * low[3];

      tmp[i][j + M/2] = img[i][jj]     * high[0] +
                        img[i][jj + 1] * high[1] +
                        img[i][jj + 2] * high[2] +
                        img[i][jj + 3] * high[3];
  }
  for (i=0, ii=0; ii<N; i++, ii +=2)
    for (j=0; j<M; j++)
    {
      img[i][j]= tmp[ii][j]   *low[0]+tmp[ii+1][j]*low[1] +
                 tmp[ii+2][j] *low[2]+tmp[ii+3][j]*low[3];

      img[i+N/2][j]= tmp[ii][j]  *high[0]+tmp[ii+1][j]*high[1] +
                     tmp[ii+2][j]*high[2]+tmp[ii+3][j]*high[3];
    }
}
```

Fig. 4. The 2D DWT using the Daub-4 for an image of size $N \times M$.

The equation above is minimized when the denominator of the second term of the inequality has the maximum value. The maximum value is achieved when a reconfigurable processor with the highest speedup is employed. We can rewrite Eq. (7) as follows:

$$t_{min} > \frac{1}{p}\left\{\sum_{i=1}^{n} M_i \frac{t_{GPP}}{max\{k_j\}}\right\} = \frac{1}{k_{max}p}\left\{\sum_{i=1}^{n} M_i t_{GPP}\right\} = \frac{1}{k_{max}p} t_{non\text{-}col}. \quad (9)$$

This equation shows the lower bound for the execution time in a collaborative system is k_{max} times less than the execution time in a non-collaborative system, where k_{max} is maximum speedup of a reconfigurable processor over a GPP. Using Eqs. (4) and (9) the lower and upper bounds of the performance is expressed using the following equation.

$$\frac{1}{k_{max}p} t_{non\text{-}col} < t_{col} \leq t_{non\text{-}col}. \quad (10)$$

4. Multimedia kernels

In this section, we explain the two chosen compute-intensive multimedia kernels namely: the discrete wavelet transform and the co-occurrence matrix. We selected these multimedia kernels because they are compute-intensive [20].

4.1. Discrete wavelet transform

The digital wavelet representation of a discrete signal X consisting of N samples can be calculated by convolving X with the low-pass and high-pass filters and downsampling the output results by a factor of 2, in such a way the two frequency bands each contain $N/2$ samples. With the correct choice of filters, this operation is reversible. This process decomposes the original image into two sub-bands: the lower and the higher band [20,21]. This transform can be extended to multiple dimensions by using separable filters. A 2D DWT can be performed by first performing a 1D DWT on each row (*horizontal filtering*) of the image followed by a 1D DWT on each column (*vertical filtering*). The digital wavelet transform is mainly used for image and video compression. Standards such as JPEG2000 [22] are based on the 2D DWT. For our investigation we selected the Daubechies' transform with four coefficients (Daub-4) [23]. Fig. 4 depicts the C implementation of the Daub-4 transform.

The DWT is much more computationally intensive than other functions of the JPEG2000 standard. For example, the obtained results in [24] show that the DWT consumes on average 46% of the encoding time for lossless compression. For lossy compression, the DWT even requires 68% of the total encoding time on average.

4.2. Co-occurrence matrix

Texture features are usually used in image and video processing. Texture features determine the dependencies between neighboring pixels within a region of interest in an image [25]. The co-occurrence matrix captures second order gray level information, which is a well known tool for texture analysis. Haralick, et al., [26] have defined some texture features which use co-occurrence matrices. The features are related to neighboring pixels at different directions and distance. The number of occurrences of two neighboring pixels with a distance d and with a certain direction is stored in a co-occurrence matrix. The co-occurrence matrix is always a square matrix of size $N_{gl} \times N_{gl}$, where N_{gl} is the number of available gray levels in the image. Consequently, the size of this matrix is independent from distance and direction neighboring pixels and also from the image size. The co-occurrence matrix consists of relative frequencies $P(i, j; d, \delta)$ of two neighboring pixels i, j

```
void Co_occurrenceMatrix()
{
 for(i=1; i < N-1; i++)
   for (j=1; j < M-1; j++)
    {
      GLCM[ img[i][j] ][ img[i-1][j-1] ]++;
      GLCM[ img[i][j] ][ img[i-1][j] ]++;
      GLCM[ img[i][j] ][ img[i-1][j+1] ]++;
      GLCM[ img[i][j] ][ img[i][j-1] ]++;
      GLCM[ img[i][j] ][ img[i][j+1] ]++;
      GLCM[ img[i][j] ][ img[i+1][j-1] ]++;
      GLCM[ img[i][j] ][ img[i+1][j] ]++;
      GLCM[ img[i][j] ][ img[i+1][j+1] ]++;
    }
}
```

Fig. 5. C implementation of the co-occurrence matrix.

separated by distance d at orientation δ in an image. In some image processing, for example medical images Haralick texture features are computed for some Region Of Interest (ROI). The size of the ROIs is not fixed and it is dependent on the input image. As many ROIs are selected, therefore for each ROI a co-occurrence matrix is computed.

There are eight directions, while only four of them are unique. The co-occurrence matrix will be symmetric if all directions are considered. Fig. 5 depicts the C implementation of the co-occurrence matrix on considering eight different directions.

Haralick's texture feature calculation can be divided into two parts. First is the calculation of co-occurrence matrices and second is the calculation of texture features using the calculated co-occurrence matrices. These steps are very time consuming. For example, for an image of size 5000 × 5000, time required is approximately 350 s using a Pentium 4 machine. The calculation of the co-occurrence matrices and Haralick texture features take 75% and 19% of the total time, respectively [24].

5. Simulation environment and tools

In this section, we present our simulation environment. We investigated multimedia kernels as a case study on a network that includes primitives to construct a backbone of reconfigurable processors in a grid network. In these primitives, different processing elements collaborate together to execute a multimedia kernel. Each processing element can be a GPP or a reconfigurable processor (Reconfigurable Element (RE)). The specification of processing elements is defined in the form of Million Instructions Per Second (MIPS) for the Standard Performance Evaluation Corporation (SPEC) benchmark. In this paper, we utilized 30, 35, 40, and 50 MIPS for the processing elements.

In order to understand how many instructions are required to execute the discussed multimedia kernels, we executed both the 2D DWT and co-occurrence matrix kernels using the SimpleScalar toolset [27] for an image of size 1024 × 1024. The number of committed instructions for the 2D DWT kernel is 46 160 416, while the number of committed instructions for the co-occurrence kernel is 83 899 404. This means that in order to provide each value for the first decomposition level of 2D DWT, 44 instructions should be processed, while for the co-occurrence kernel 80 instructions should be processed.

The simulation environment is an extended version of the GridSim (a traditional Java-based discrete-event grid simulator) [28,29]. We configured and prepared the GriSim simulator based on the multimedia kernel properties to support the collaborative processing between reconfigurable processors. This extension of the GridSim is called CRGridSim. In CRGridsim, each application can be broken down into different subtasks called *gridlets*. Each application is packaged as gridlets whose contents include the task length in Millions of Instructions (MI). To simplify the simulation of the proposed approach, we assumed that reconfigurable processors do not support partial reconfiguration.

Table 2
Specifications of the simulated environment.

Parameter	Value
Maximum packet size	32 and 64 KBytes
User–router bandwidth	100 Mb/s
Router–router bandwidth	1000 Mb/s
Number of images	40 (Table 3)
Number of users	1
Size of images	Based on Table 3
PE specification (MIPS)	Based on Table 4
Speedup for RE	10 in compared to a GPP
Reconfiguration file size	3 Mb
Reconfiguration speed	3 Mb/s
Reconfiguration time	1 s
Number of bits per pixel	24 bit

Table 3
Images and their corresponding gridlet specifications for different multimedia kernels.

Image			# of instructions (MIPS)	
Group	Size	# of images	2D DWT	Co-occ. matrix
1	768 × 1024	10	35	64
2	1024 × 1024	10	46	84
3	1200 × 1600	10	84	156
4	2134 × 2848	10	267	493

Table 4
The specification of processing elements in terms of MIPS.

Processing elements	MIPS
Main GPP	30
Collaborator 1 (GPP or RE)	35
Collaborator 2 (GPP or RE)	50
Collaborator 3 (GPP or RE)	40

The multimedia kernels were simulated using the primitive in Fig. 2(a) with 2 and 3 collaborator processing elements. The specifications of the simulated environment and primitives are depicted in Table 2. The speedup of a single reconfigurable processor over a single GPP for multimedia kernels was set to 10. This is because, as we mentioned in Section 2 the average speedup for the multimedia kernels for a single reconfigurable processor was measured as $(4.75 + 7.3 + 20)/3 = 10.68$.

The images' size and specification of the processing elements are presented in Tables 3 and 4, respectively. Four groups of different images with various sizes have been collected in Table 3. Each image is sent in the uncompressed form to the processing elements. The required instructions to process each image is packed as a gridlet and is submitted to the related processing elements, either a GPP or an RE. Table 4 depicts the specification of processing elements in terms of MIPS.

It should be noted that for the simulated topology, the Routing Information Protocol (RIP) is utilized by the simulator. The arbitrary number of collaborators can be used that depends on the application requirements and available processing elements. However, increasing the number of processing elements increases the overhead, for example, the communication time. In order to map the multimedia kernels on reconfigurable processors, the possible reconfigurable parameters should be identified. For example, variable wavelet filter lengths and variable wavelet decomposition levels are two reconfigurable parameters for DWT. Different steps to execute an application on the CRGC are depicted in Fig. 6. In Fig. 6,

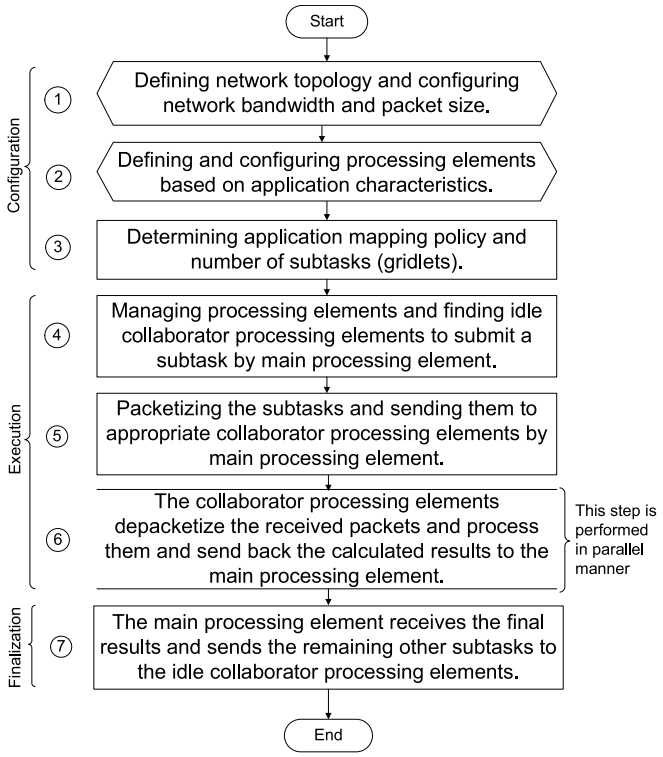

Fig. 6. The flowchart of application execution in CRGridsim simulator.

box 1 is managed by the NetUser module, box 2 is managed by CRGridsim (the RE, PE, and NetUser modules), boxes 3 and 4 are managed by CRGridsim (NetUser module), and boxes 5 and 6 are managed by the main processing element (i.e., the requesting processing element) in CRGridsim.

In the first three steps, the network topology and application mapping policy are defined and parameters such as network bandwidth, packet size, and number of gridlets (subtasks) in the simulator are configured. Steps 4, 5, and 6 show the execution of gridlets by the collaborator processing elements. Available collaborator processing elements are selected to execute a set of gridlets, therefore, the main processing element packetizes each gridlet and sends to related collaborator processing elements. The target collaborator processing element collects the received packets and executes the gridlets and finally sends back results to the main processing element. In step 7, the main processing element receives the calculated results and sends the remaining gridlets to the collaborator processing elements.

6. Simulation results

In this section, we present the simulation results that were obtained using the CRGridSim simulator [18]. First, we present the application mapping on the CRGC for multimedia kernels. Subsequently, the performance evaluation of the CRGC is presented.

6.1. Application mapping on CRGC

Parallel architectures can be programmed using two models of parallel programming, data parallelism and task parallelism. In data parallelism, data is partitioned and distributed among the processing elements. For example, in the case of image processing applications, images are split into several sub-images and each sub-image is processed by a processing element. In task parallelism, the instructions can be grouped into tasks and each task is assigned to a processing element. In other words, the task is split into a number of subtasks and each subtask is assigned to a specific processor. In addition, the data necessary for each subtask is sent to the appropriate processing element.

Based on the simulation environment presented in Section 5, we map our applications on two different configurations. The first configuration is the collaboration of GPPs with a packet size of 64 KBytes. The second configuration is the collaboration of reconfigurable processors (elements) with a packet size of 64 KBytes. Tables 5 and 6 show the mapping of the 2D DWT on the first and second configurations, respectively. GPP0 is the main processing element, while other processors are the collaborator processing elements. Columns three to six represent the number of assigned gridlets from each group (Table 3) to each processing element. For example, in Table 5, GPP1 processes 3, 0, 2, and 2 gridlets from groups 1, 2, 3, and 4, respectively. The seventh column shows the total number of processed gridlets by each processing element. The last column represents the total number of executed instructions by each processing element. For instance, in Table 5, GPP1 executes 3×35, 0×46, 2×84, and 2×267 MIs for groups 1, 2, 3, and 4, respectively. The total executed instructions (see Table 3) are 807 MIs (million of instructions).

Three collaborator processing elements in Table 5 process $7 + 8 + 10 = 25$ gridlets, while the main processing element processes 15 gridlets. It should be noted that these numbers are obtained by the executing of a simulation program presented in Section 5. In other words, the GPP0 processes the most number of gridlets. On the other hand, three collaboration reconfigurable processors in Table 6 process $10 + 10 + 11 = 31$ gridlets, while the main processing element processes 9 gridlets, which is the least number of gridlets compared to other processing elements. As a result, the collaboration of reconfigurable processors can process more gridlets than the collaboration of GPPs.

In order to increase the computational time, both media kernels are integrated and executed together. Table 7 depicts the mapping of the execution of both media kernels on the collaboration of GPPs, while Table 8 depicts the mapping of the execution of both media kernels on the collaboration of reconfigurable processors. Three collaborator processing elements in Table 7 process $8 + 13 + 10 = 31$ gridlets, while the main processing element processes 9 gridlets. On the other hand, three collaboration reconfigurable processors in Table 8 process $12 + 13 + 12 = 37$ gridlets, while the main processing element processes 3 gridlets. As a result, in the collaboration mode, each collaborator processing element processes much more gridlets than the main GPP.

6.2. Performance evaluation

In order to evaluate the proposed approach, we considered two packet sizes, 32 KBytes and 64 KBytes (the largest packet sizes in the networks). Our results show that using larger packet sizes lead to higher performance than smaller packet sizes. Larger packet sizes decreases the communication overhead due to sending less packets. In our case, we have 40 images with a total size of 294 MBytes. These images have 4539 packets of 64 KBytes and 9219 packets of 32 KBytes. In addition, we considered four different configurations, collaboration of 3 GPPs, a GPP with 2 REs, 4 GPPs, and a GPP with 3 REs. The configuration of 3 GPPs means that 2 GPPs are collaborating with a main GPP. The star topology has been used for the collaboration mechanism that a structure of this topology was depicted in Fig. 2(a).

Fig. 7 depicts the speedups of the first two configurations, 3 GPPs and a GPP with 2 REs over a GPP for different packet sizes. Fig. 8 also depicts the speedups of the last two configurations, 4 GPPs and a GPP with 3 REs, over a GPP for different packet sizes. Our observations from these figures are the following. First, increasing the packet size from 32 KBytes to 64 KBytes improves the

Table 5
Application mapping of the 2D DWT on collaboration of GPPs on a grid computing.

Resource		Gridlets (based on Table 3)					Total # of ins. for PEs
Type	MIPS	# of assigned gridlets from group 1	# of assigned gridlets from group 2	# of assigned gridlets from group 3	# of assigned gridlets from group 4	Total # of assigned gridlets	
GPP0	30	2	5	4	4	$2+5+4+4=15$	1704
GPP1	35	3	0	2	2	$3+0+2+2=7$	807
GPP2	50	2	2	1	3	$2+2+1+3=8$	1047
GPP3	40	3	3	3	1	$3+3+3+1=10$	762

Table 6
Application mapping of the 2D DWT on collaboration of reconfigurable processors (elements) on a grid computing.

Resource		Gridlets (based on Table 3)					Total # of ins. for PEs
Type	MIPS	# of assigned gridlets from group 1	# of assigned gridlets from group 2	# of assigned gridlets from group 3	# of assigned gridlets from group 4	Total # of assigned gridlets	
GPP0	30	2	2	2	3	$2+2+2+3=9$	1131
RE1	35	3	3	3	1	$1+3+3+3=10$	762
RE2	50	1	3	3	3	$1+3+3+3=10$	1226
RE3	40	4	2	2	3	$4+2+2+3=11$	1201

Table 7
Application mapping of the 2D DWT+co-occurrence matrix on collaboration of GPPs on a grid computing.

Resource		Gridlets (based on Table 3)					Total # of ins. for PEs
Type	MIPS	# of assigned gridlets from group 1	# of assigned gridlets from group 2	# of assigned gridlets from group 3	# of assigned gridlets from group 4	Total # of assigned gridlets	
GPP0	30	2	3	2	2	$2+3+2+2=9$	2588
GPP1	35	2	2	1	3	$2+2+1+3=8$	2978
GPP2	50	2	4	5	2	$2+4+5+2=13$	3438
GPP3	40	4	1	2	3	$4+1+2+3=10$	3290

Table 8
Application mapping of the 2D DWT+co-occurrence matrix on collaboration of reconfigurable processors (elements) on a grid computing.

Resource		Gridlets (based on Table 3)					Total # of ins. for PEs
Type	MIPS	# of assigned gridlets from group 1	# of assigned gridlets from group 2	# of assigned gridlets from group 3	# of assigned gridlets from group 4	Total # of assigned gridlets	
GPP0	30	0	0	1	2	$0+0+1+2=3$	1760
RE1	35	4	4	2	2	$4+4+2+2=12$	2916
RE2	50	3	3	3	4	$3+3+3+4=13$	4447
RE3	40	3	3	4	2	$3+3+4+2=12$	3167

performance. As can be seen, the speedups for the packet size 64 KB are larger than the speedups for packet size 32 KB. This is due to the fact that larger packet sizes decrease the communication overhead. Second, the collaboration of reconfigurable processors improve the performance more than the collaboration of GPPs. For our configuration, the performance improvement of the collaboration of reconfigurable processors over the collaboration of GPPs is up to 2.5. This is because, based on the specifications of the processing elements in the simulation environment in Tables 2 and 4 each reconfigurable processor processes more instructions than a GPP. Finally, executing computationally intensive applications yields much more performance than non-computationally intensive applications. This is because the impact of the communication overhead is reduced compared to the computational time. As it can be seen in Figs. 7 and 8, combination of both kernels increases the computational time and then it obtains more speedups than the execution of each kernel separately. In other words, the speedup of both kernels together (2D DWT+co-occurrence matrix) is more than the speedup of 2D DWT and co-occurrence matrix and the speedup of the co-occurrence matrix is more than 2D DWT. This is because of the number of instructions related to each kernel (see Table 4). Additionally, increasing the number of collaborator processing elements improves the performance. This is because the submitted subtasks to each collaborator are decreased. This reduces the number of processed instructions by each processing element. Furthermore, from Fig. 8(a) and (b) it can be observed

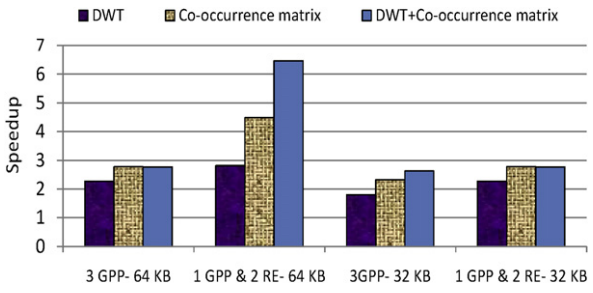

Fig. 7. Speedup for different configurations with 2 collaborator processing elements over one GPP.

when the processing power of RE changes from 2 to 10 the performance of the system is increased.

It should be noted that the theoretical upper bound of speedup that can be obtained in the CRGC approach is 30× using Eq. (10), while the actual speedup is 7.2*x*. The reason for this difference in the speedup is due to the impact of the following factors: communication time, reconfiguration time, application mapping policy, and load balancing. To achieve higher speedup in CRGC, it should be noted to select computationally intensive applications with minimum data transfer and communication overheads. In addition, in

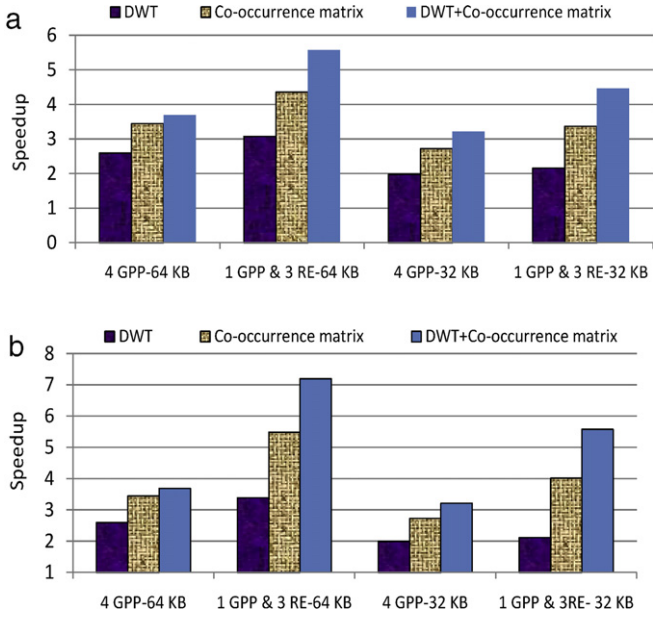

Fig. 8. Speedup for different configurations with 3 collaborator processing elements over one GPP. (a) When reconfigurable elements are 2 times faster than a GPP. (b) When reconfigurable elements are 10 times faster than a GPP.

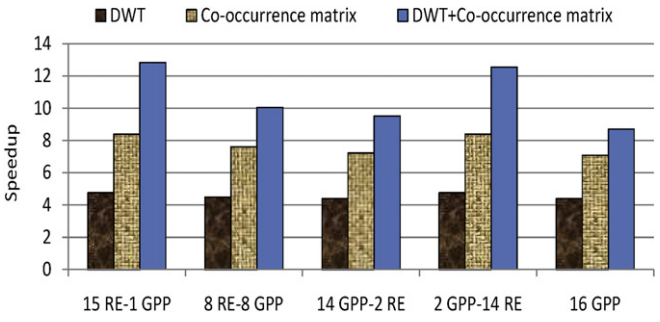

Fig. 9. Speedup of CRGC with different numbers of processing elements over one GPP with packet size 64 KB.

a real network, each processing element can collaborate with a limited number of neighbor processing elements. Increasing the number of neighbor processing elements increases the communication time and this reduces the performance. The speedup of the CRGC with different numbers of processing elements for 100 gridlets is depicted in Fig. 9.

In this figure, the speedup of 5 different configurations to highlight the role of number and type of processing elements over a GPP is evaluated. These configurations are: 15 RE-1 GPP, 8 RE-8 GPP, 2 RE-14 GPP, 14 RE-2 GPP and 16 GPP. From this figure, the following facts can be observed:

1. The configuration of 15 RE-1 GPP has the highest speedup. This is because this configuration uses more reconfigurable processors than GPPs.
2. Increasing the number of processing elements (GPPs or reconfigurable processors) in the CRGC system increases the speedup. When the number of processing elements is increased the number of assigned gridlets to each processing element is decreased resulting in the execution time decreasing.
3. Processing both 2D DWT and co-occurrence matrix kernels together achieves more speedups than processing them individually. This is due to the fact that the computational time will be significant compared to the communication overhead time.

7. Conclusions

In order to achieve both flexibility and high-performance, we proposed the collaboration of reconfigurable processing elements in grid computing. Reconfigurable computing provides much more flexibility than application-specific integrated circuits and much more performance than GPPs. Grid computing increases the performance of computationally intensive applications by exploiting the parallelism. Each part of an application can be executed on a processing element of a grid.

In this paper, we proposed the concept of Collaborative Reconfigurable Grid Computing (CRGC). In CRGC reconfigurable processors capabilities are utilized in the grid environment. The collaboration was implemented using a neighborhood policy. In this policy, each processing element requests assistance only from its neighboring processing elements. In addition, we analyzed the lower and upper bounds of performance for the CRGC. We simulated the collaboration of four processing elements with different configurations on the grid computing. This environment was simulated using a CRGridSim simulator. Subsequently, we mapped several compute-intensive multimedia kernels on the simulated environment using realistic assumptions. The experimental results showed that the collaboration of reconfigurable processors improves performance by up to 2.5x compared to the collaboration of GPPs. The results show that the proposed CRGC approach is capable of improving the performance of compute-intensive multimedia kernels more than non-compute-intensive kernels due to the communication overhead. Furthermore, the CRGC is more beneficial for the compute-intensive application with lower data communication overheads such as scientific and mathematical problems.

References

[1] C. Bobda, R. Hartenstein, Introduction to Reconfigurable Computing Architectures, Algorithms, and Applications, Springer, 2007.
[2] K. Compton, S. Hauck, Reconfigurable computing: a survey of systems and software, ACM Computing Survey 34 (2) (2002) 171–210.
[3] R. Luethy, C. Hoover, Hardware and software systems for accelerating common bioinformatics sequence analysis algorithms, Drug Discovery Today: BIOSILICO 2 (1) (2004) 12–17.
[4] S. Vassiliadis, S. Wong, G.N. Gaydadjiev, K.L.M. Bertels, G.K. Kuzmanov, E. Moscu Panainte, The Molen polymorphic processor, IEEE Transactions on Computers (2004) 1363–1375.
[5] G.K. Kuzmanov, G.N. Gaydadjiev, S. Vassiliadis, The molen media processor: design and evaluation, in: Proc. Int. Workshop on Application Specific Processors, 2005, pp. 26–33.
[6] J.P. Morrison, P.D. Healy, P.J. O'Dowd, Architecture and implementation of a distributed reconfigurable metacomputer, in: Proc. 2th Int. Symp. on Parallel and Distributed Computing, 2003, pp. 153–158.
[7] F.M. Vallina, E. Oruklu, J. Saniie, Distributed processing network architecture for reconfigurable computing, in: IEEE Proc. Int. Conf. Electro-Information Technology, vol. 3, 2005, pp. 6–12.
[8] M. Smith, G.D. Peterson, Parallel application performance on shared high-performance reconfigurable computing resources, Performance Evaluation 60 (1–4) (2005) 107–125.
[9] K.D. Underwood, R.R. Sass, W.B. Ligon, Acceleration of a 2D-FFT on an adaptable computing cluster, in: 9th Annual IEEE Symp. on Field-Programmable Custom Computing Machines, FFCM-01, 2001, pp. 180–189.
[10] M.A. Tahir, A. Bouridane, F. Kurugollu, A. Amira, Accelerating the computation of GLCM and haralick texture features on reconfigurable hardware, in: Proc. of the Int. Conf. on Image Processing, 2004, pp. 2857–2860.
[11] D.K. Iakovidis, D.E. Maroulis, D.G. Bariamis, FPGA architecture for fast parallel computation of co-occurrence matrices, Microprocessors and Microsystems 31 (2007) 160–165.
[12] T.C. Chiang, P.Y. Wu, Y.M. Huang, A limited flooding scheme for mobile ad hoc networks, in: Proc. IEEE Int. Conf. on Wireless And Mobile Computing, Networking And Communications, August 2005, pp. 473–478.
[13] T. Guan, E. Zaluska, D.D. Roure, A grid service infrastructure for mobile devices, in: Proc. of First Int. Conf. on Semantics, Knowledge and Grid(SKG-05), 2005, pp. 42–46.

[14] TraGrid, Purdue teragrid resources, August 2010. http://www.rcac.purdue.edu/teragrid/resources/.
[15] DAS-3, Distributed asci supercomputer: Das-3, August 2010. http://www.starplane.org/das3/.
[16] DAS-4, Distributed asci supercomputer: Das-4, August 2010. http://wiki.cs.vu.nl/das-4/index.php/Main_Page.
[17] M. Ahmadi, A. Shahbahrami, S. Wong, Collaboration of reconfigurable processors in grid computing for multimedia kernels, in: Proc. of the 5th Int. Conf. on Grid and Pervasive Computing, May 2010, pp. 5–14.
[18] S. Wong, M. Ahmadi, Reconfigurable architectures in collaborative grid computing: an approach, in: Proc. 2nd Int. Conf. on Networks for Grid Applications, 2008.
[19] S. Finch, Transitive Relations, Topologies and Partial Orders, 2003. http://algo.inria.fr/bsolve/.
[20] A. Shahbahrami, M. Ahmadi, S. Wong, K.L.M. Bertels, A new approach to implement discrete wavelet transform using collaboration of reconfigurable elements, in: Proc. of Int. Conf. on ReConFigurable Computing and FPGAs, 2009, pp. 344–349.
[21] E.J. Stollnitz, T.D. Derose, D.H. Salesin, Wavelets for Computer Graphics: Theory and Applications, Morgan Kaufmann, 1996.
[22] M. Rabbani, R. Joshi, An overview of the JPEG2000 still image compression standard, Signal Processing: Image Communication 17 (1) (2002) 3–48.
[23] M.A. Trenas, J. Lopez, E.L. Zapata, F. Arguello, A memory system supporting the efficient SIMD computation of the two dimensional DWT, in: Proc. IEEE Int. Conf. on Acoustics Speech and Signal Processing, vol. 3, May 1998, pp. 1521–1524.
[24] A. Shahbahrami, B. Juurlink, S. Vassiliadis, Implementing the 2D wavelet transform on SIMD-enhanced general-purpose processors, IEEE Transactions on Multimedia 10 (1) (2008) 43–51.
[25] R.W. Conners, C.A. Harlow, Theoretical comparison of texture algorithms, IEEE Transactions on Pattern Analysis and Machine Intelligence 2 (3) (1980) 204–222.
[26] R.M. Haralick, K. Shanmugam, I. Dinstein, Textural features for image classification, IEEE Transactions on Systems, Man, and Cybernetics 3 (6) (1973) 610–621.
[27] T. Austin, E. Larson, D. Ernst, SimpleScalar: an infrastructure for computer system modeling, IEEE Computer 35 (2) (2002) 59–67.
[28] R. Buyya, M.M. Murshed, GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, Concurrency and Computation: Practice and Experience 14 (13–15) (2002) 1175–1220.
[29] A. Sulistio, G. Poduval, R. Buyya, C.K. Tham, On incorporating differentiated levels of network service into gridsim, Future Generation Computer Systems 23 (4) (2007) 606–615.

Mahmood Ahmadi received the B.S. Degree in Computer Engineering from Isfahan University, Isfahan, Iran in 1995. He received the M.Sc. Degrees in computer architecture and engineering from Tehran Polytechnique University, Tehran, Iran in 1998. From 1999 to 2005, he was a faculty member at Razi University in Kermanshah in Iran. In October 2005, he joined the Faculty of Electrical Engineering, Mathematics, and Computer Science (EEMCS), Delft University of Technology, Delft, The Netherlands, as a full-time Ph.D. student. He got his Ph.D. in May 2010. His research interests include computer architecture, network processing, signal processing, and reconfigurable computing. He is currently working as an assistant professor in the Department of Computer Engineering at the Razi University of Kermanshah. He is a member of the IEEE, and HIPEAC.

Asadollah Shahbahrami received the M.Sc Degree in Computer Engineering-Machine Intelligence, from Shiraz University, Shiraz, Iran, in 1996. He has worked as a member of faculty staff of Electrical Engineering at the University of Guilan, Guilan, Iran, for 7 years from 1996 to 2003. In January 2004, he joined the Faculty of Electrical Engineering, Mathematics, and Computer Science (EEMCS) at Delft University of Technology, The Netherlands, as a full time Ph.D. student under the advisors Prof. Stamatis Vassiliadis and Dr. Ben Juurlink. He got his Ph.D. degree in September 2008. His research interests include computer architecture, image and video processing, multimedia instructions set design, and SIMD programming. He is currently working as an assistant professor in the Department of Computer Engineering at the Uiversity of Guilan. He is a member of the IEEE, ACM, and MVIP.

Stephan Wong received his Ph.D. degree in December, 2002 from the Electrical Engineering Department at Delft University of Technology (TU Delft), The Netherlands. He is currently working as an assistant professor in the Computer Engineering Laboratory at TU Delft. He has considerable experience in the design of embedded media processors. He has also worked on microcoded FPGA complex instruction engines and the modeling of parallel processor communication networks. His research interests include embedded systems, multimedia processors, parallel/distributed processing, microcoded machines, and network processors. He is a member of the IEEE, HIPEAC, and ACM.