Improving Utilization of Reconfigurable Resources Using Two Dimensional Compaction

Ahmed A. El Farag Hatem M. El-Boghdadi* Samir I. Shaheen
Computer Eng. Dept., Faculty of Eng.,
Cairo University, Giza, Egypt.

Abstract

Partial reconfiguration allows parts of the reconfigurable chip area to be configured without affecting the rest of the chip. This allows placement of tasks at run time on the reconfigurable chip. Area management is a very important issue which highly affect the utilization of the chip and hence the performance.

This paper focuses on a major aspect of moving running tasks to free space for new incoming tasks (compaction). We study the effect of compacting running tasks to free more contiguous space on the system performance. First, we introduce a straightforward compaction strategy called Blind compaction. We use its performance as a reference to measure the performance of other compaction algorithms. Then we propose a two-dimensional compaction algorithm called one-corner compaction. This algorithm runs with respect to one chip corner. We further extend this algorithm to the four corners of the chip and introduce the 4-corner compaction algorithm. Finally, we compare the performance of these algorithms with some existing compaction strategies [3]. The simulation results show improvement in average task allocation time when using the 4-corner compaction algorithm by 15% and in chip utilization by 16% over the Blind compaction. These results outperform the existing strategies.

1. INTRODUCTION

Today's Field-Programmable Gate Array (FPGA) devices provide several millions of gates and allow partial reconfiguration. Partial reconfiguration allows part of the chip to be configured with a new task without affecting other currently running tasks. While this technique can increase the device utilization, it also leads to complex allocation situations for dynamic task sets. Even if good placement method is used, the reconfigured area will be fragmented. A high fragmentation can lead to undesirable situation where a new task cannot be placed although there would be sufficient number of scattered cells [1]. So, there exist a serious need for well-defined system service that helps to efficiently manage the area resource. The benefit of using the area management strategy is to increase the area utilization and so increase the system performance [2]. On the other hand, these benefits are paid for by overheads in the required computation time of the area management system and task relocation. Here we adopt task compaction as our strategy to task relocation.

The area management contains two aspects: task placement and task compaction. A task placement is responsible for the decision where a task is mapped in the reconfigurable area. If placement has failed, the compaction unit tries to rearrange the running tasks to reduce chip fragmentation and then find a place for the incoming task.

Compaction is collecting used area near to each other and so collecting scattered free cells to generate larger free areas. Hopefully, the new free areas can accommodate the new incoming task. Compaction requires preemption, where running tasks are stopped and continued at a different location [2]. If all the running tasks are involved in the compaction process, then the compaction is total other wise it is partial.

Total compaction is moving all tasks, if possible, to free more contiguous area. This can be done in two ways. The first way is to stop all tasks, compact the tasks and then resume all tasks again. The second way is to stop one task and resume it in a new location. This process is repeated for all tasks. The former method gives better arrangement but consumes much more time. The later method consumes minimum stopping time per task, while the stopping time of a task is only the time to move its configuration to the new place. But this method needs more effort in the choice of the selected task and its new place. In this paper we concentrate on the second method.

Partial compaction is moving some of the tasks (one or more) on the chip to generate a free location area that just fits the new task. This method preempts tasks that only help in the solution. The method does not disturb other tasks, but it consumes more time to test all possible task movements to find the solution [3]. Also, this method presents a short-term solution for the current new task, and does not help any other incoming task or improve the fragmentation of the reconfigurable area.

In this paper, we discuss a preemptive partially reconfigurable system that executes a set of independent tasks. We study different compaction strategies and show their effects on FPGA area utilization and performance. We compare the quality and complexity of these strategies. First, a base line compaction algorithm, *Blind Compaction,* is presented. The algorithm is a straightforward algorithm that we introduce to be able to evaluate other compaction

* Author for Correspondence: helboghdadi@eng.cu.edu.eg

978-3-9810801-2-4/DATE07 © 2007 EDAA

algorithm. The algorithm is one- dimensional (compaction of tasks is done in one direction) and the order of tasks is preserved after compaction. Second, we compare the existing one-dimensional partial compaction algorithms [3] with the new base line. Then, we introduce a two-dimensional compaction algorithm, *one-corner compaction*, in which we compact the tasks in two dimensions toward one corner of the chip. Finally, we extended the corner compaction algorithm to work with respect to the four corners of the chip, *4-corner compaction*. In this algorithm, tasks that are closer to a certain chip corner are compacted toward that corner using the corner compaction algorithm. Our results show an improvement of about 16% in area utilization and 15% in allocation time over the blind compaction.

The next section presents the previous work. Section 3 describes the system model. In Section 4, we describe the Blind compaction algorithm. Section 5 describes the corner and 4-corner compaction algorithms. Section 6 presents the simulation results. In Section 7, we summarize our results and make some concluding remarks.

2. RELATED WORK

A lot of work has been done in offline arrangement of tasks, where task sizes and service times are known in advance. In an offline scenario, one can afford to spend the time to derive optimal or near-optimal solutions. In this case the problem is much similar to the traditional packing problem [4].

A substantial work has been done in finding empty place on the reconfigurable area [5] and the online task placement methodology [6][7] to improve their efficiency. They all deal with non-preemptive tasks, and so they do not offer task re-arrangement. Compton *et al.* [8] discuss a hardware modification to the FPGA that provides task relocation and transforms to reduce fragmentation. Task transforms consist of a series of rotation and flip operations.

Diessel *et al.* [3] tackle the fragmentation problem in partially reconfigurable FPGAs. They perform a task rearrangement by techniques denoted as local repacking and ordered compaction. Local repacking method attempts to repack the tasks within a part of the chip so as to accommodate the waiting task as well. A quad tree decomposition of the free space in the chip is used and a depth-first search of the tree allows promising parts to be identified and evaluated. This operation requires $O(n^3 \log n)$ where n is the number of currently running tasks on the chip. Diessel *et al.* [3] also presented an ordered compaction heuristic that moves some tasks to one side of the chip and places the waiting task at the freed location. Ordered compaction therefore has the effect of moving the running tasks that are to be compacted closer together while preserving their relative order. To select the moved tasks a direct dependency graph is built and depth-first traversal is applied with some candidate cells to check the minimum cost movement place. This operation requires $O(n^3)$ where n is the number of currently running tasks on the chip. We compare our work with the results of their ordered compaction method.

Koester [9] assumes one dimensional compaction on the FPGA area; this is due to the block distribution of the used reconfigurable device.

In contrast to the previous work, this paper focuses on pre-emptive task re-arrangement when placement unit could not find free contiguous space for the new incoming task. We assume that tasks need a rectangular area. We further discuss the compaction algorithms and compare their performances. This paper presents a novel two-dimensional compaction algorithm and its extension that improve the reconfigurable chip utilization and the system performance.

3. SYSTEM MODEL

This section presents the tasks characteristics for a partial reconfigurable system. Then, we define the preemptive system model we use in this paper.

3.1 Task Characteristics

An $H \times W$ partially reconfigurable FPGA chip consists of H rows and W columns is used. The lower left corner is the chip origin. Part of the chip can be configured without affecting the rest of the chip. Our system assumes that tasks arrive online, queued and placed in arrival order. Task parameters (size, arrival time, service time) are not known in advance. These task parameters are defined as follows. For a task t_i, $t_i = (h_i, w_i, x_i, y_i)$, where h_i (resp. w_i) represents its height (resp. width) and measured in number of rows (resp. columns). The parameters h_i and w_i are uniformly distributed in a predefined region. The size of the task is $h_i \times w_i$. The rectangular area assigned to the task is presented by its lower left corner (x_i,y_i) where x_i: row number, and y_i: column number.

Tasks are re-locatable; i.e. the task can be placed at any free area in the reconfigurable chip area, and at any time it can be suspended and resumed in another free place. Re-locatable tasks can be placed at arbitrary positions with different row and column offsets. In fact, task relocation involves some difficulties, while intra-task wires are translated, wires running between different tasks or between tasks and I/Os need to be re-routed dynamically. Here, we assume that tasks are independent, so we do not have onboard task communications.

The asynchronous tasks arrival times as well as the tasks service times are uniformly distributed in a predefined interval and are a-priori unknown. These characteristics reflect a general-purpose computing system.

Formally, a task t_i arrives at time a_i , starts to execute at time s_i , and finishes at f_i . Thus, the task's response time is

given by *resp* $(t_i) = f_i - a_i$. The allocation time, *alloc* (t_i) is the time the task is waiting at the top of the waiting queue till it finds a place on the reconfigurable area.

All arriving tasks are queued in arrival order. Tasks are taken one by one from this list to be located in the reconfigurable chip.

3.2 System Characteristics

In this paper we consider homogeneous devices. Although many new reconfigurable devices are heterogeneous, the homogeneous systems are still wide spread in many applications. Also, the work presented here could be applied to the homogeneous portions of the heterogeneous devices.

The resource management system consists of two main units: Placement unit and Compaction unit. The placement unit is responsible of finding empty place on the FPGA to place the task, while the compaction unit is responsible of performing the compaction algorithm to free contiguous place for the new task.

When a task executed and there exist some waiting tasks in the waiting list, the above process is repeated. This process continues till all the required tasks end execution.

3.3 Performance Measures

The main objective is to improve chip utilization and system performance. For a task set T with N tasks used in the evaluation process, we test the reconfigurable chip utilization, $U(T)$, that quantifies how well we use the resource. With respect to time analysis of the system, we measure the average allocation time, $A(T)$, and the average response time, $R(T)$. The allocation time quantifies the average waiting time for each task while testing the FPGA area till an empty place exists. The response time quantifies the average time duration for each task from entering to leaving the system. We also calculate the average number of parallel running tasks on the chip, n, and the number of compaction trials done to allocate all tasks. The overall execution time, $E(T)$, of all tasks is the time elapsed from the arrival of the first task till the departure of the last task. Assuming that first task arrives at time 0, we can calculate the above measures as follows:

$$E(T) = \max(f_i) \quad \text{where } (1 \le i \le N),$$

$$A(T) = \frac{1}{N}\sum_{i=1}^{N} alloc(t_i) \ ,$$

$$R(T) = \frac{1}{N}\sum_{i=1}^{N} resp(t_i) \ ,$$

$$U(T) = \frac{\sum_{i=1}^{N} h_i \times w_i \times (f_i - s_i)}{E(T) \times H \times W} \times 100$$

4. BLIND COMPACTION ALGORITHM

In this section, we introduce a baseline for all compaction algorithms and call it Blind compaction. The baseline algorithm is a straightforward methodology to compact tasks and its performance can be used as a reference for other compaction algorithms. The algorithm is blind in that it performs the compaction if a place is not found even if the compaction will not result in an enough area for the incoming task. The algorithm is one-dimensional, in which, compaction of tasks are done in one direction. There is no loss of generality to choose the chip right side.

Definition 1 For $H \times W$ reconfigurable area with n active tasks, a task $t_j = (h_j, w_j, x_j, y_j)$, $1 \le j \le n$, is in the *direct east* of a task $t_i = (h_i, w_i, x_i, y_i)$, $1 \le i \le n$, if $y_j \ge y_i + w_i$ and there exists a row k, such that $x_i \le k < x_i + w_i$ and $x_j \le k < x_j + w_j$ where $1 \le k \le H$. ■

The above definition means that task t_j is in direct east of task t_i if t_j is on the right of t_i and they have a common row. For example in Figure 1, row 6 is common between t_3 and t_4, also t_4 is on the right of t_3. Thus t_4 is in direct east of t_3. While t_2 is not in direct east of t_3, and t_3 is not in direct east of t_1.

The Blind compaction algorithm simply moves all tasks to the nearest place to the right side of the chip. Tasks are sorted in ascending order with respect to the distance between task's right edge and the chip right side. The first task is selected and moved to the right such that its right edge is directly touching the chip right side. The next task is selected and moved to the most possible right free place till it touched either the chip right side or any other task's left edge. The process is repeated till all tasks have been visited. In doing this, each task moves once to its final place. Also, each task is stopped the minimum possible time (the time to transfer the task data from the current location to the new location). Finally tasks are compacted while preserving their relative ordered.

Figure 2 shows the Blind compaction algorithm. The first step calculates the distance m_i between the right side of each task and the chip right edges. Step 2 sorts the tasks with respect to the distances computed in Step 1. Any sorting algorithm can be used, for example a linear sort with complexity $O(n^2)$. The second step selects a task in order and computes the free distance between this task and the nearest task to its right. The second step takes $O(n^2)$. The overall complexity of this method is $O(n^2)$. Since the tasks are sorted, no task will need further movement after it reaches its destination. As intended, this Blind compaction algorithm will be used as a reference for other compaction algorithms. ■

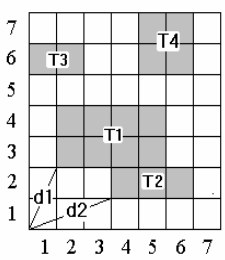

Figure 1: Relative distance.

Algorithm 1: Blind Compaction (L, L′)
Input: A list L with n active tasks,
$$L = \{t_i; t_i = (h_i, w_i, x_i, y_i), 1 \le i \le n\}$$
Output: A compacted list L' with n active tasks,
$$L' = \{t_i; t_i = (h_i, w_i, x_i, y_i'), 1 \le i \le n\}$$
1. For i = 1 to n
$m_i = W - (y_i + w_i)$
2. Sort L with respect to m_i in descending order, in L'.
3. For i = 1 to n
  3.1 $t_i = L'(i)$
  3.2 For j = 1 to i-1
    $t_j = L'(j)$
    If (t_j) in the *direct east* of (t_i)
      then $m_i = \min(m_i, y_j - (y_i + w_i))$
    Next j
  3.3 Move t_i to the right m_i columns, i.e. $(y'_i = y_i + m_i)$
 Next i

Figure 2: Blind compaction Algorithm.

5. NEW COMPACTION ALGORITHMS

In this section we introduce a two-dimensional compaction algorithm. Instead of compacting tasks in one direction towards the chip edge, the two-dimensional compaction algorithm moves tasks towards one chip corner. A task t_i located at (x_i, y_i) could change its position after compaction to (x_i', y_i') where $x_i \neq x_i'$ and $y_i \neq y_i'$. The problem of choosing the tasks' moving order arises. Section 5.1 identifies how we arrange tasks to choose the moving order. Section 5.2 describes our two-dimensional compaction algorithm. In Section 5.3, we modify the algorithm in Section 5.2 to compact tasks towards the four corners of the chip.

5.1 Task arrangement

Since we are trying to move the active tasks towards a corner, we need to define the order of tasks to be moved. In the Blind compaction algorithm, we use the distance between the right side of each task and the right edge of the chip as the criteria to sort tasks. In two-dimensional environment assuming we are moving tasks to the bottom-left corner, the distance from a task's bottom-left corner to the bottom left corner of the chip, which can be calculated as $d_i = \sqrt{x_i^2 + y_i^2}$ for a task t_i, might not yield always a good choice. As shown in Figure 1, for tasks t_1 and t_2, although $d_2 > d_1$, we need to move t_2 before t_1.

Definition 2 For $H \times W$ reconfigurable area with n active tasks, a task $t_j = (h_j, w_j, x_j, y_j)$, $1 \le j \le n$, is in the *south-west region* of a task $t_i = (h_i, w_i, x_i, y_i)$, $1 \le i \le n$, if $y_j < y_i + w_i$ and $x_j < x_i + h_i$. ■

The above definition means that task t_j is in the *direct south west* of task t_i if the lower left corner of task t_j is in the south-west region of task t_i. For example, in Figure 1, t_1, t_2, *and* t_3 are in the south-west region of t_4.

Definition 3 A task t_j has *relative distance,* Xt_j, less than the *relative distance,* Xt_i, of a task t_i if:
(1) t_j is in the *south-west region* of t_i or
(2) $d_j < d_i$ and (1) is not true. ■

The above definition defines the criteria on which we will arrange the tasks to be moved in the two-dimensional compaction algorithm. For example, in Figure 1, $Xt_2 < Xt_1$ because t_2 is in the *south-west region* of t_1. Also $Xt_2 < Xt_3$ because $d_2 < d_3$ and t_2 is not in the *south-west region* of t_3.

The algorithm in Figure 3 takes $O(n^2)$ where each task has to ask the other n tasks to identify its relative order in the sorted list *SL*. For example in Figure 1, since $Xt_2 < (Xt_1, Xt_3, Xt_4)$, the first task is t_2. The sorted list of tasks in Figure 1 will be $\{t_2, t_1, t_3, t_4\}$. ■

5.2 Two-dimensional compaction

In the two-dimensional compaction, we move tasks in both vertical and horizontal directions to one of the chip corners (south-west, south-east, north-east, or north-west). Without loss of generality, we use the bottom-left corner. The order of the tasks is taken as their relative order with respect to the selected corner as shown in the previous section. The compaction process is shown in Figure 4.

Algorithm 2: Arrange (*L*,*SL*)
Input: Set of active tasks $L = \{t_i, 1 \le i \le n\}$
Output: Sorted list of active tasks with respect to
the *relative distance*, $SL = \{t_i, 1 \le i \le n\}$
1. Get first element from L and add in the empty list SL
2. For j = 2 to n
 2.1 $t_j = L(j)$
 2.2 For i = 1 to j-1
  $t_i = SL(i)$
  If $Xt_j < Xt_i$
   then add (t_j) before (t_i) in *SL* , *and* go to step 3
  Next i
 2.3 Add (t_j) at the end of *SL*
3- Next j

Figure 3: Arrange Algorithm.

Algorithm 3: One-Corner-Compaction(L, L')

Input: List of active tasks,
$L = \{ t_i = (h_i, w_i, x_i, y_i) \ ; 1 \le i \le n \}$
Output: List of active tasks with new positions,
$L' = \{ t_i = (h_i, w_i, x_i', y_i') \ ; 1 \le i \le n \}$

1- Call Arrange (L, L`)
2- For $i = 1$ to n
2.1 $t_i = L'(i)$
2.2 Find new location (x, y) for t_i
2.3 If $((x < x_i)$ and $(y \le y_i))$ or $((x \le x_i)$ and $(y < y_i))$
then $x_i' = x$, $y_i' = y$
Next i

Figure 4: Two-dimensional compaction Algorithm.

In Figure 4, Step 1 takes $O(n^2)$. In Step 2, finding new location for task t_i takes $O(n^2)$. Thus, the total algorithm requires $O(n^2)$ to complete the compaction process.

5.3 Modified two-dimensional compaction

The new idea here is to use the four corners, not only one corner (bottom left corner), to compact tasks towards them. This algorithm highly improves area fragmentation and so highly improves the system performance. This algorithm has the same complexity of the corner compaction algorithm. The additional step is to divide the tasks with respect to corners, which requires only $O(n)$. Thus, the total complexity of this algorithm is $O(n^2)$. The performance of this method is shown in the next section (this algorithm mentioned as 4-corner).

In the 4-corner compaction algorithm, the tasks that are near to any corner are compacted towards that corner. A task is nearest to certain corner if distance between task's center to that corner is shorter than other corners. The chip has four corners: north east corner, C_{ne}, north west corner, C_{nw}, south east corner, C_{se}, and south west corner, C_{sw}. All tasks are tested at first and divided to four groups, each group is compacted towards its nearest chip corner using the one-corner compaction algorithm. Relative distances are computed with respect to the selected corner.

Figure 5 shows the effect of applying different compaction algorithms. Figure 5 (a) shows the initial placement of tasks before any compaction. Figure 5 (b) shows the result of the Blind compaction algorithm, while Figure 5 (c) and (d) show the task placement after applying the one-corner and the 4-corner compaction algorithms respectively.

6. RESULTS

Several experiments were done to compare the performance of the different compaction methods. For each experiment, sets of 10,000 tasks characterized by 4 independently chosen uniformly distributed random variables were generated. Two random variables represent the two task side lengths (maximum of 32×32). Two other random variables represent the inter-task arrival period (with maximum of 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, and 300 time units) and the task service period (maximum 1000 time units). The tasks were queued in arrival order and placed in bottom left method to a simulated FPGA of size 64×64. The configuration delay per cell was fixed to 0.001 time units [3] (The effect of changing this value should be considered in a future work.) We assume that the compaction algorithms run on a host computer and are not taken into account in the task response time.

We use the blind compaction as a reference to compare compaction methods. In Figure 6, we can reach an improvement over the blind compaction in allocation time of 15% with 4-corner compaction, while with 1-corner compaction, it is about 8% and with partial compaction it is only 5%. In Figure 7, we can reach an improvement in response time over the blind compaction up to 37% with 4-corner compaction, while with 1-corner compaction, it is up to 21% and with partial compaction it is only up to 11%. In Figure 8, we can reach an improvement over the blind compaction in utilization near 16% with 4-corner compaction and between 8% and 9% with 1-corner compaction while with partial compaction it is about only 4%. In Figure 9, we compare the average number of active (working) tasks on the FPGA chip in the operation time, we can see that the 4-corner compaction can manage place for 10% active tasks at a time more than other compaction methods. This property improves the chip utilization and also reduces the total response time.

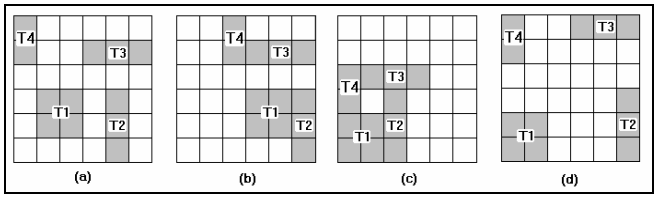

Figure 5: (a) Initial tasks (b) Blind Compaction (c) One corner (d) Four corners

7. CONCLUDING REMARKS

In this paper, we introduced the blind compaction algorithm that we considered its performance a reference for all compaction algorithms. The blind compaction represented the basic improvement that can be achieved through compaction. Then we introduced new two-dimensional compaction algorithms, one-corner compaction and 4-corner compaction, that improve the chip utilization and hence the system performance. We compared these new algorithms and the partial compaction algorithms [3], with

the reference performance. The comparison showed an improvement in the system performance. For the utilization, the 4-corner compaction is 12% better than the partial compaction and 7% better than the one-corner compaction.

We have several directions to extend this work. We plan to apply these algorithms in real time system and test the effect of compaction on the system miss ratio. Real time operating system is an important application of such systems. Extending the work to consider heterogeneous reconfigurable system is another direction. In addition to the placement of the tasks, it is necessary to consider the communication infrastructure. Therefore, the model can be extended by a formal description of the communication infrastructure.

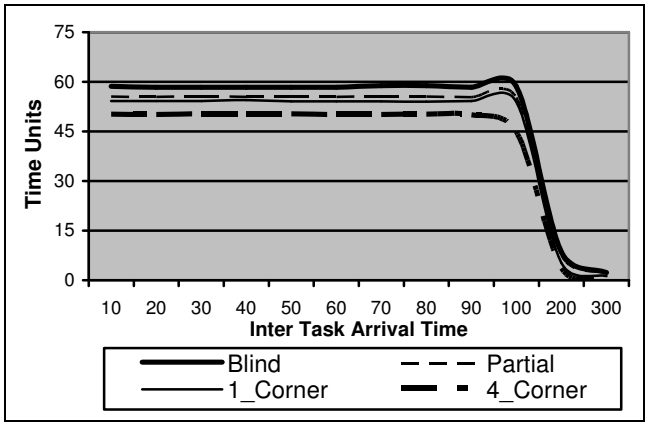

Figure 6: Compaction algorithms effect on allocation time.

REFERENCES

[1] Manuel G. Gericota, Gustavo R. Alves, Miguel L. Silva, José M. Ferreira, "On-line Defragmentation for Run-Time Partially Reconfigurable FPGAs", *FPL 2002*, LNCS 2438, pp. 302-311, 2002.

[2] Manuel G. Gericota, Gustavo R. Alves, Miguel L. Silva, José M. Ferreira, "Run-Time Management of Logic Resources on Reconfigurable Systems", *In Proceedings of the Design, Automation and Test in Europe 2003 Conference and Exhibition (DATE'2003)*, Munich, Germany, March 2003, pp. 974-979.

[3] O. Diessel, H. ElGindy, M. Middendorf, H. Schmeck, and B. Schmidt, "Dynamic scheduling of tasks on partially reconfigurable FPGAs". In *IEEE Proceedings on Computers and Digital Techniques*, volume 147, pages 181–188, May 2000.

[4] Sandor P. Fekete, and Jorg Schepers. "A Combinatorial Characterization of Higher-Dimensional Orthogonal Packing". *In Mathematics of Operations Research*, Volume 29: 353-368 (2004).

[5] Kiarash Bazargan, Ryan Kastner, and Majid Sarrafzadeh. "Fast Template Placement for Reconfigurable Computing Systems". *In IEEE Design and Test of Computers*, volume 17, pages 68–83, 2000.

[6] H. Walder, C. Steiger, and M. Platzner, "Fast Online Task Placement on FPGAs: Free Space Partitioning and 2D-Hashing", *International Parallel and Distributed Processing Symposium*, April 2003.

[7] M. Handa and R. Vemuri, "An Efficient Algorithm for Finding Empty Space for Online FPGA Placement", *Design Automation Conference*, San Diego, CA, June 2004, pp. 960-965.

[8] Katherine Compton, James Cooley, Stephen Knol, and Scott Hauck. "Configuration Relocation and Defragmentation for Reconfigurable Computing". *In Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines* (FCCM). IEEE CS Press, April 2001.

[9] H. Kalte, M. Koester, B. Kettelhoit, M. Porrmann and U. Rückert " A Comparative Study on System Approaches for Partially Reconfigurable Architectures ". *Engineering of Reconfigurable Systems and Algorithms (ERSA06)*.

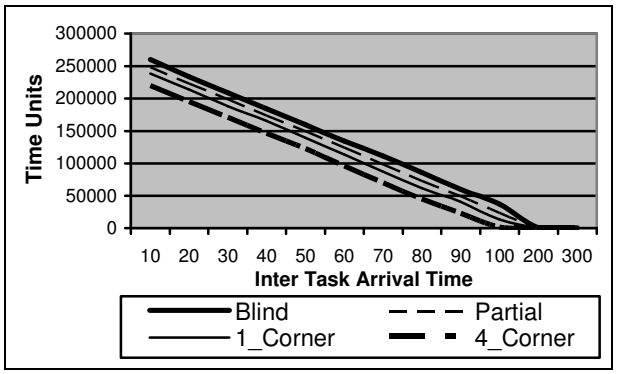

Figure 7: Compaction algorithms effect on response time.

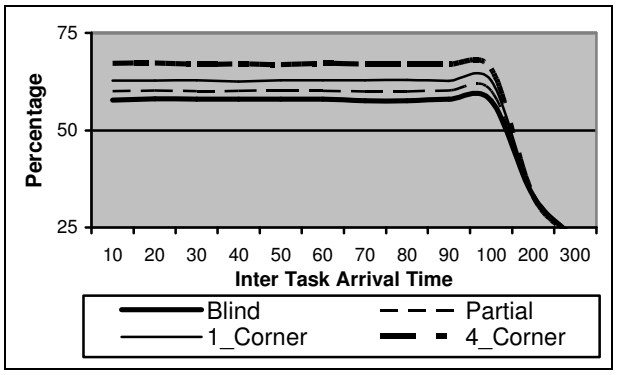

Figure 8: Compaction algorithms effect on chip Utilization.

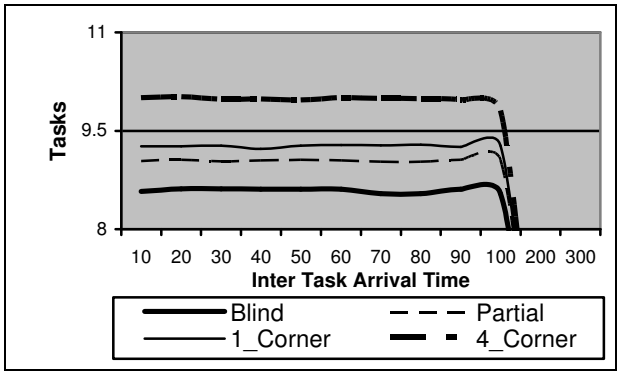

Figure 9: Average number of active tasks.