Genetic algorithms for hardware–software partitioning and optimal resource allocation

Madhura Purnaprajna *, Marek Reformat, Witold Pedrycz

Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Alberta, Canada T6G2V4

Received 7 September 2004; received in revised form 24 August 2005; accepted 3 October 2006

Available online 8 December 2006

Abstract

A scheme for time and power efficient embedded system design, using hardware and software components, is presented. Our objective is to reduce the execution time and the power consumed by the system, leading to the simultaneous multi-objective minimization of time and power. The goal of suitably partitioning the system into hardware and software components is achieved using Genetic Algorithms (GA). Multiple tests were conducted to confirm the consistency of the results obtained and the versatile nature of the objective functions. An enhanced resource constrained scheduling algorithm is used to determine the system performance. To emulate the characteristics of practical systems, the influence of inter-processor communication is examined. The suitability of introducing a reconfigurable hardware resource over pre-configured hardware is explored for the same objectives. The distinct difference in the task to resource mapping with the variation in design objective is studied. Further, the procedure to allocate optimal number of resources based on the design objective is proposed. The implementation is constrained for power and time individually, with GA being used to arrive at the resource count to suit the objective. The results obtained are compared by varying the time and power constraints. The test environment is developed using randomly generated task graphs. Exhaustive sets of tests are performed on the set design objectives to validate the proposed solution.

© 2006 Elsevier B.V. All rights reserved.

Keywords: Hardware–software partitioning; Genetic algorithms; Reconfiguration

1. Introduction

System design involves intelligent management of available resources to meet performance requirements, while achieving the desired functionality. Design partitioning is governed by a trade-off between a number of system performance parameters such as time, area, weight, cost, power, reliability and flexibility. More specifically, embedded system requirement necessitates the use of time and power efficient designs. In addition to these requirements the flexibility of the design in terms of time to design, configure and reconfigure the system is also a major deciding factor. Hardware–software co-design involves managing these issues using hardware and software system resources. The optimization process involved in hardware–software co-design may be broadly divided into two steps.

* Corresponding author.
E-mail address: madhura@ece.ualberta.ca (M. Purnaprajna).
Firstly, it determines the number of resources suitable for a given design objective and secondly, it partitions the system into hardware and software elements and improves the solution based on quality of this partition. The efficiency of the partitioned system is graded in terms of its performance/watt ratio. Low power design strategies are incorporated to prolong battery life, reduce packaging and cooling costs for mobile applications and operations at high speeds. In conventional microprocessors a high-speed operation often results in high power consumption, which is undesirable at system level. Also, using a pure Field Programmable Gate Array (FPGA) based design results in high power consumption in the system. Our objective is to optimize the system for both power and time considerations. Area and cost restrictions are reflected in terms of resource constraints. The simultaneous optimization of power and time is addressed, which has not been widely discussed previously. The proposed scheduling algorithm manages both scheduling and allocation simultaneously based on task priority. A test environment was developed to analyze and validate these design objectives. The test graphs described, represent the system architecture at task-level granularity [1], which depicts a generalized description from a software or a hardware perspective. Section 2 provides a brief correlation between the scheduling techniques in heterogeneous computing, high-level synthesis and hardware–software co-design. This section also provides a background on the ongoing research activities in the field of hardware–software co-design. Section 3 illustrates the basic issues in hardware–software co-design. Section 4 provides the problem formulation. Section 5 discusses the application of Genetic Algorithms (GA) to the optimization problem of hardware software partitioning. The test suite and the experimental set-up to implement the algorithms are described in Section 6. Sections 9 and 10 provide the detailed description of the experiments and the results, respectively, and conclusions are drawn in Section 11.

2. Background

Hardware–software co-design comprises of optimal scheduling and allocation of heterogeneous resources. Scheduling and allocation has been addressed in operations research, heterogeneous computing, multi-processor scheduling and high-level synthesis. Heterogeneous computing employs a collection of machines with varying architectures, which perform computationally diverse applications. The scheduling procedure performs a task to machine mapping for the sequence of tasks with the intention of minimizing time and cost or both. A comparative study is made using 11 heuristics to optimize the system implementation in [2]. This concept resembles the approach of mapping an operation to a resource in hardware–software partitioning. In high-level synthesis, the problem of partitioning and scheduling resembles the case of multi-processor scheduling, which is known to be an NP-complete problem [3]. Scheduling determines the exact time instance at which a task is executed. It is classified on the basis of its constraints, such as time and resources. In time constrained scheduling, bound by a hard-deadline, the algorithm computes the optimal combination of resources required to meet this timing requirement. In resource constrained scheduling, with a restriction in the number of resources available for the task computation, scheduling decides the best time instance to execute a task, so as to minimize the total computation time. The As Soon As Possible (ASAP) and the As Late As Possible (ALAP) algorithms compute task schedules for unconstrained graphs. They are primarily used to determine the upper and lower bounds for the schedule times [4]. A number of heuristics have been explored to optimize the process of design partitioning in hardware–software co-design [5–7]. Many of these heuristics aim at multi-objective optimization, with conflicting objectives for hardware and software design units [8–12]. A comparison to these existing techniques is difficult due to the absence of well-defined benchmarks in terms of architecture and functionality, for hardware–software co-designs. However, the concept of replicable randomly generated task-graphs, used in this paper, encourages future comparisons [1]. This representation is independent of the target architecture. In [7] the heuristics applied are Simulated Annealing, Tabu Search and Genetic Algorithms. Genetic algorithms have been used to optimize the design for speed, subject to area constraints. Reconfigurable hardware resources with finite reconfiguration times [13] have been used. The objective function measures the speed improvement achieved using hardware–software co-design over an all-software implementation. Partial reconfiguration in hardware has also been used to enhance resource utilization [14,15]. In [14], the multi-objective criterion of time and the
energy-delay product is minimized. Another study compares partitioning of knowledge-based systems and circuit-partitioning using Simulated Annealing, Kernighan & Lin and Hierarchical clustering [10]. With the introduction of hardware elements in an all-software implementation, there is an overall improvement in the speed of operation [11]. Repetitive operations, such as loops in a task graph are executed on hardware. Predominantly, most power optimization techniques focus on fabrication-related issues, such as voltage scaling [16], reducing clock speeds and introducing gated clocks [17]. MOGAC [8] aims at multi-objective partitioning criteria for optimization in power and price, with time constraints. Power calculation is based on pre-defined values for peak power dissipation, idle power consumption and power efficiency for each hardware resource. This calculation is based on obtaining energy and ignores the additional power consumed due to the switching activity between them. Also, the GA formulation requires a complicated technique of clustering and validation to obtain only valid structures. MOGAC does not assume any limitation in the number of hardware or software elements in its architecture. This is quite contradictory to practical applications. In our discussion, for time optimization the quality of partition is determined by the total time required for task execution based on the cumulative latencies at the nodes. The additional overheads account for the inherent delays incurred at edges of the graph. For power, it is the summation of the power consumed by each of the graph nodes. In addition, the inter-processor communication overheads add to the switching activity in the system. These overheads are incurred only in heterogeneous multi-processor systems. In time constrained scheduling, with a set hard-deadline, the resources required to satisfy these timing constraints constitute the target architecture for the system implementation. Resource allocation and utilization has been only partially addressed in the case of hardware–software co-design. In [18], experiments are carried out to compare the improvement in system performance by increasing the number of dynamically reconfigurable logic elements and varying the reconfiguration time. However, there is no established method for determining the optimal number of resources required for a given system based on the design objective. Most co-design approaches consider resource-constrained architectures. Such architectures may have under-utilized resources which add to idle power and unused area in turn affecting the system efficiency.

3. Essential issues in co-design

The computational sequence of tasks in an application is represented by Control and Data flow graphs (CDFG). For a System on a Chip (SoC) application, these tasks can be distributed both on hardware as well as software resources. Resources termed as hardware can be Application Specific Integrated Circuits (ASIC) or a reconfigurable device, such as a Field Programmable Gate Array (FPGA). The devices commonly referred to as ‘software’ are General-Purpose Processors (GPP) or Digital Signal Processors (DSP). On account of customization in hardware, ASICs have a very large design time, but have a significantly high performance. However, ASICs are not suitable on account of their enormous design times and lack of reconfigurability. FPGAs, on the other hand, are readily reconfigurable devices and hence chosen as ‘hardware’ in our architecture. These devices play a significant role in co-design to enhance the resource utilization of the system on account of their reconfigurable capabilities. With the advent of runtime reconfiguration, FPGAs provide the ability to adapt to the design requirement. This stands a significant advantage compared to customized ASICs. Although they are time-efficient, FPGAs are expensive in terms of area and power. Also, introduction of reconfiguration adds time and power overheads. In the absence of runtime reconfiguration, the FPGA architecture is limited to its pre-defined fixed configuration. FPGAs provide task execution at higher speeds than most GPPs, but are not power-efficient. GPPs are relatively less time-efficient but provide a low-cost, low power additional resource to explore parallelism, as compared to FPGAs. Also, they are not restricted by their configuration options since there is little or no reconfiguration overhead. A slow speed and low power GPP is chosen as ‘software’ in the architecture under consideration.

Fig. 1 shows the relative variations in flexibility and speed. Allocating all the tasks on the same hardware resource for speed enhancement, results in complicated scheduling procedures. Similarly forcing all the tasks on software overshoots the time deadline. Consequently, it is efficient to implement a design on multiple processors (FPGAs and GPPs), to utilize parallelism. The system block diagram is shown in Fig. 2. The architecture under consideration
comprises of both pre-configured and reconfigurable hardware resources on an FPGA, software resources on GPPs and a shared Dual port memory (Dual Port RAM). The reconfigurable hardware resource allows partial reconfiguration at run-time. The software resource does not require any reconfiguration time to adapt itself to the system resource requirement. The dual port memory allows independent access to both the hardware and software resources. The main purpose of this architectural approach is to avoid bus-contention and arbitration for memory access. This is in contrast to other architecture previously considered in [15]. The application to be executed on this architecture comprises of a sequence of tasks with two functionally varying tasks. There exists a fundamental difference in the task execution in the two types of resources. FPGAs accommodate concurrent execution of tasks, limited by the availability of resources on the device. Number of tasks that can be executed concurrently on an FPGA depends on the number of system units that can be configured on the device. This is not applicable in software, since tasks mapped to software are executed sequentially. Due to the absence of concurrency in software, the number of tasks that can be executed at any given time is restricted to the number of GPPs available. Also, the GPP is assumed to consume low power as compared to the FPGA. Consequently, with respect to execution times, it is assumed that a task implemented in software takes greater number of clock cycles than that on hardware. Partial run-time reconfiguration on FPGAs, allows part of the device to be reconfigured while rest of the device functions normally [19]. Introduction of partial run-time reconfigurable capabilities increases resource utilization by allowing reconfiguration of the unused resources to adapt according to the system resource requirement, encountered during scheduling. However, reconfiguration involves an additional cost of time and power. Optimal resource allocation allows efficient utilization of system resources. The architecture under consideration has resources with varying system parameters. The utilization ratios of these resources vary with design objectives. Hence for a given design objective, GA is used to

Table 1

<table>
<thead>
<tr>
<th>No.</th>
<th>All-hardware Type1 = 4 Type2 = 4 Time Power</th>
<th>All-hardware Type1 = 6 Type2 = 2 Time Power</th>
<th>All-software Time Power</th>
<th>Hardware–software Time Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>312 1185</td>
<td>307 1185</td>
<td>980 447</td>
<td>433 732</td>
</tr>
<tr>
<td>2</td>
<td>314 1191</td>
<td>313 1191</td>
<td>948 447</td>
<td>419 733</td>
</tr>
<tr>
<td>3</td>
<td>317 1170</td>
<td>313 1191</td>
<td>948 447</td>
<td>426 720</td>
</tr>
<tr>
<td>4</td>
<td>357 1175</td>
<td>337 1175</td>
<td>1219 444</td>
<td>436 720</td>
</tr>
<tr>
<td>5</td>
<td>307 1221</td>
<td>322 1221</td>
<td>947 454</td>
<td>407 751</td>
</tr>
</tbody>
</table>
determine the type and number of resources suited to meet the design requirements. Optimal resource allocation mainly avoids resources which are unused or under-utilized in an application. This reduces the idle power consumption in unused resources and also the additional area occupied by resources that are not used efficiently.

Table 1 summarizes the relative difference in execution times for a design implemented on all-hardware, all-software and hardware–software co-design platforms. As observed, an all-hardware implementation results in the best solution when the objective is entirely to minimize time. For power optimization, an all-software solution is best suited as it results in the least power consumption. A co-design approach, which is a combination of the two, is best suited for the multi-objective optimization in power and time.

4. Problem description

The objective of our approach is to propose a multi-objective optimization procedure to achieve hardware software partitioning that is independent of the target implementation. The only information about the target architecture is given in the form of a list of available resources. The implementation architecture suggests the optimal target architecture, based on the available resources. With a set number of resources available, GA is used to optimize the system performance in terms of the execution time, power consumed and the multi-objective optimization of time and power. The optimization is achieved by maximizing a function representing the objectives called fitness function:

\[
\text{Maximize}(\text{Fitness})
\]  

The following sections describe the fitness function used for the above mentioned design objectives.

4.1. Time optimization

To minimize the total time taken, the fitness function \( f_{\text{time}} \), is defined as

\[
f_{\text{time}} = \frac{1}{1 + T_{\text{norm}}}
\]  

where \( T_{\text{norm}} \) is the normalized time and is given by:

\[
T_{\text{norm}} = \frac{T_{\text{total}}}{T_{\text{max}}}
\]  

where \( T_{\text{total}} \) is the time at which the last task in the priority list completes execution. \( T_{\text{max}} \) is the maximum schedule time for a given task graph. In this context, the execution time is the maximum when all the tasks are mapped to software. \( T_{\text{total}} \) is used to assess the quality of the partition. The inherent latencies for the resource is the execution time for a task on hardware (\( t_h \)), software (\( t_s \)) and reconfigurable hardware (\( t_r \)). Apart from the inherent latencies of the design units, additional delay is consumed in a multiprocessor environment. This has been integrated to emulate a practical system. The additional time incurred accounts for the delays due to inter-processor communication times at the edges of the task nodes. The fitness function for time optimization, with the additional communication delay is given by:

\[
f_{\text{ctime}} = \frac{1}{1 + T_{\text{norm}}} * \frac{1}{1 + C_{\text{norm}}}
\]  

where \( C_{\text{norm}} \) is the normalized communication delay and is given by:

\[
C_{\text{norm}} = \frac{C_{\text{delay}}}{C_{\text{delaymax}}}
\]  

where \( C_{\text{delay}} \) is the total communication delay due to hardware–hardware (\( t_{hh} \)), hardware–software (\( t_{hs} \)), software–hardware (\( t_{sh} \)) or software–software (\( t_{ss} \)) communication overheads. \( C_{\text{delaymax}} \) is the maximum communication delay. This is experienced when the delay due to communication is the most for each possible case. This is when there is hardware to software communication, or vice versa, at every operation.

4.2. Power optimization

The fitness function for optimization in power, \( f_{\text{power}} \), is expressed as

\[
f_{\text{power}} = \frac{1}{1 + P_{\text{norm}}}
\]  

where \( P_{\text{norm}} \) is the normalized Power and is given by:

\[
P_{\text{norm}} = \frac{P_{\text{total}}}{P_{\text{max}}}
\]  

where \( P_{\text{total}} \) is the sum of power consumed by all the elements in the task graph, with the given partition. \( P_{\text{max}} \) is the maximum power that can be consumed for a given task graph. Power rating for each of the node depends on the type of resource and the
type of operation. They are, power rating for hardware \((P_h)\), software \((P_s)\) and reconfigurable hardware \((P_r)\). Maximum power is consumed when all the tasks are implemented on hardware (or reconfigurable hardware), since hardware resources consume higher power compared to software. In the second case maximum power is consumed when all the tasks are implemented on reconfigurable hardware, with reconfiguration required at each node. In a multiprocessor environment additional power consumed is due to the switching activity in the system on account variation in resource allocation between consecutive tasks. For power optimization:

\[
f_{\text{power}} = \frac{1}{1 + P_{\text{norm}}} * \frac{1}{1 + S_{\text{norm}}}
\]

where \(S_{\text{norm}}\) is the normalized switching activity and is given by:

\[
S_{\text{norm}} = \frac{S_{\text{total}}}{S_{\text{max}}}
\]

where \(S_{\text{total}}\) is the total switching power due to inter-processor communication. \(S_{\text{max}}\) is the maximum switching power experienced. Maximum switching power is consumed when tasks alternate between hardware and software resource allocation. This results in memory access at every node. They are hardware–software \((P_{hs})\) and software–hardware \((P_{sh})\) interactions, resulting in the switching activity at the memory interface bus. The hardware–hardware \((P_{hh})\) and software–software \((P_{ss})\) interaction avoids these overheads. Here, hardware represents both pre-configured and reconfigurable hardware.

4.3. Time and Power optimization

For the multi-objective optimization in Power and Time the fitness function, \(f_{\text{PandT}}\), is defined as:

\[
f_{\text{PandT}} = f_{\text{time}} * f_{\text{power}}
\]

where \(f_{\text{time}}\) and \(f_{\text{power}}\) are given by Eqs. (2) and (6), respectively.

For simultaneous optimization in power and time with the additional overheads of communication delays and switching power, the fitness function is given by:

\[
f_{\text{PandTAdd}} = f_{\text{time}} * f_{\text{power}}
\]

where \(f_{\text{time}}\) and \(f_{\text{power}}\) are defined in Eqs. (4) and (8), respectively. The fitness function attempts to compete with two conflicting objectives. The time optimization attempts to minimize the total time by forcing all operations to hardware (and reconfigurable hardware), where as the power-aware system tries to avoid hardware resources. These orthogonal objectives result in hardware–software solutions, with hardware resources in time critical paths. To obtain a balance between both the objectives, the fitness function for the multi-objective optimization is the product of the two fitness functions. This results in simultaneous minimization of the two objectives.

4.4. Optimization for resource allocation

This problem of hardware software partitioning is extended to allocate optimal number of resources to proportionally allocate resources with high utilization and reduce the number of under-utilized extra resources. Design considerations limit the numbers of resources that can be used by the given application. These limitations exist on account for area and cost constraints. Hence it is appropriate to perform task scheduling based on resource availability. However, among the resources allocated, not all are effectively utilized. GA is used to arrive at the best number of resources of each type. The architecture under consideration comprises of three types of resources, namely, pre-configured hardware, reconfigurable hardware and software. Resource allocation is effective when all the resources allocated are completely utilized. The objective of resource allocation is to arrive at the optimal combination of resources to be used to satisfy the design objective. Two penalty functions have been introduced to satisfy hard requirements regarding time and power. These requirements cannot be exceeded, during the optimization procedure.

4.4.1. Penalty for Time

The fitness function adds penalty for violating a deadline. This deadline is set based on the ASAP-Time calculated for the task graph

\[
P_t = \begin{cases} 
0, & \text{if } T \leq D \\
\alpha \ast (T - D), & \text{otherwise}
\end{cases}
\]

where \(T\) is the time taken for the execution of a task graph. \(D\) is the hard-deadline set for a task graph. \(\alpha\) is a constant.

4.4.2. Penalty for Power

The fitness function adds penalty when the maximum power consumed in the system exceeds the set
requirement. This limitation is set based on the maximum power consumed by the task graph

\[ P_1 = \begin{cases} 0, & \text{if } P \leq P_{\text{set}} \\ \beta \cdot (P - P_{\text{set}}), & \text{otherwise} \end{cases} \]  

(13)

where \( P \) is the power evaluation for the given combination of resources. \( P_{\text{set}} \) is the set maximum power limitation of the system. \( \beta \) is a constant.

4.4.3. Penalty for resource utilization
An additional penalty is added to limit the number of resources used to avoid unused resources. This directs the search process to avoid under-utilization of resources. The Penalty function is given by:

\[ P_2 = \begin{cases} 1, & \text{if } T_{\text{Unused}} = 0 \\ T_{\text{Unused}}, & \text{otherwise} \end{cases} \]

(14)

where \( T_{\text{Unused}} \) is the total number of unused resources.

\[ T_{\text{Unused}} = S_{\text{Unused}} + R_{\text{Unused}} + H_{1\text{Unused}} + H_{2\text{Unused}} \]

\[ S_{\text{Unused}} = \text{Unused software.} \]

\[ R_{\text{Unused}} = \text{Unused reconfigurable hardware.} \]

\[ H_{1\text{Unused}} = \text{Unused pre-configured hardware type1.} \]

\[ H_{2\text{Unused}} = \text{Unused pre-configured hardware type2.} \]

The resulting fitness function which meets the required deadline with optimal resource allocation while optimizing power is given by:

\[ f_{\text{allocate}} = f_{\text{calc}} \cdot \frac{1}{1 + P_1} \cdot \frac{1}{1 + P_2} \]  

(15)

where \( f_{\text{calc}} \) represents objective as defined earlier in Eqs. (4), (8) and (11).

5. Genetic Algorithms

5.1. Optimization framework

GA is a powerful and widely used stochastic search based algorithm [20]. It is an effective technique to solve combinatorial optimization problems, which are known to be non-deterministic in nature and are associated with a large combination of feasible solution space or search space. It has been demonstrated that GA is effective in avoiding the local optimal solution and achieves results close to the global optima. Fig. 3 is the logical flow for the implementation of GAs. The basic components in GA formulation may be described as follows [21].

5.1.1. Phenotype to genotype
Representation is a key issue in the GA algorithm. The possible solutions to the problem under consideration are encoded as a string of finite length, referred to as a chromosome.

5.1.2. Initial population
GA works on processing a set of probable solutions. This first set of potential solutions is called the initial population.

5.1.3. Evaluation function
The quality of each chromosome is assessed by an evaluation function. This determines the fitness of the chromosome.

5.1.4. Reproduction
Based on the fitness of the individual, the next generations of possible solutions are created in the process of reproduction. Stochastic transformations are introduced to form new individuals. This probabilistic approach creates a wide space of possible solutions. Two main operators for reproduction are crossover and mutation. Crossover creates new individuals by copying parts of two other individuals. Mutation introduces random transformations to the existing chromosome and creates a new individual.

5.1.5. Termination criteria
The algorithm terminates when the fitness function stabilizes after iterating over a predetermined number of generations.
5.2. Partitioning

The mapping of the partitioning problem to genetic algorithms is described as follows.

5.2.1. Chromosomal representation

The total number of tasks (nodes) in the task graph determines the length of the chromosome. A binary chromosome represents a system to classify tasks between hardware and software, since there are two types of possible resources. A task implemented on hardware is represented by ‘1’ and that on software is represented by ‘0’. A chromosome with trinary values is used to classify tasks on hardware, reconfigurable hardware and software. A ‘0’ represents software, ‘1’ represents hardware with fixed configuration and a ‘2’ represents reconfigurable hardware, classifying tasks between software, reconfigurable hardware and fixed pre-configured hardware. For optimizing resource allocation additional fields tracking the number of resources of each type are appended to each chromosome. The number of types of resource determines the number of additional alleles. The value of the allele varies from a minimum of one resource to the maximum number of resource available for that particular type of resource. With this chromosomal representation, all possible combinations that can be represented are valid. Hence, there is no further process of validation required even after any manipulation of the chromosomes in terms of crossover and mutation.

5.2.2. Initial population

The GA partitioning tool starts with random assignments for the alleles. This assignment randomly partitions the system into hardware and software. For the case of resource optimization, a random number (within the range specified) is assigned.

5.2.3. Fitness function

Partitioning determines the type of resource allocated to each task. The quality of the partition is determined by the scheduling operation (described in the next section). For optimization in time the total execution time determines the fitness. The total execution time is the time at which the last task completes execution. For power optimization, the total power consumed by all the resources determines the fitness. The product of the two fitness functions (time and power) is used for simultaneous power and time optimization. With the introduction of inter-processor communication overhead, the additional time and power incurred on account of multi-processor allocation is calculated. The time overhead is on account of memory–FPGA and memory–GPP communication, which is absent in case of all-hardware and all-software design approaches. Similarly, the power overhead in the system is on account of the additional switching activity between memory and the processors. Further, reconfiguration requires additional time and power to reconfigure the resource. A resource is reconfigured, when the configuration of the reconfigurable resource does not match that of the task to be scheduled. Once reconfigured, the actual execution time on a reconfigured hardware unit is the same as its pre-configured counterpart. Hence, it is preferred to minimize the number of reconfiguration and utilize the configuration for the maximum number of tasks. In the case of resource allocation, the tasks are scheduled by varying the number of resources. In this process the number of resources allocated is forced to be retained to the minimum possible.

5.2.4. Reproduction

A single point crossover is adapted. Mutation operation, in case of partitioning, moves a task randomly from hardware to software or reconfigurable hardware, or vice versa. For resource allocation, mutation introduces a random number of resources. These operations are decided based on parameters termed as probability of crossover ($P_c$) and probability of mutation ($P_m$). The selection procedure randomly chooses between Roulette wheel [20] and Tournament type of selection. Reproduction retains the best chromosome in each generation, using the elitist selection criteria [20].

5.2.5. Termination criteria

GA terminates after executing a predetermined number of iterations (generations). It is seen that, towards the end, the fitness function begins to stabilize.

5.3. Scheduling

An resource constrained scheduling algorithm is proposed. This is based on prioritizing tasks, explained in the following section.

5.3.1. Criteria for setting task priority

The earliest and the latest possible time instance, at which a task can be scheduled without any resource constraints, are termed as the AsapTime
and the AlapTime, respectively. These are set using ASAP and ALAP algorithms. Since these algorithms are applicable only to unconstrained resource scheduling, these values define the number of tasks occurring before and after a given task, in the sequence of tasks in our experiments. Initial nodes of a task graph are assumed to start at time zero. The finish time for a task is given by the sum of the start time and the execution time for that operation based on the type of resource allocated. The actual start time for each task is determined by the finish time of the preceding task. The task with the least (zero) number of preceding tasks is assigned the highest priority, since there are no preceding tasks that need to be performed prior to this task. The priority is based on the readiness of the task to be executed. The slack determined as the difference between AsapTime and AlapTime resolves the priority between tasks with the same start times.

5.3.2. Resource constrained scheduling

After the task priority is determined, the tasks are scheduled based on resource availability. A task is postponed when the assigned resource is in use by some preceding task. On completion of the assigned task, the resource is released for reuse by the proceeding tasks, with a lower priority. Incase of reconfigurable hardware, the execution time is decided based on its configuration and the configuration of the assigned task. In case of mismatch the resource may be reconfigured. Additional time and power is consumed to reconfigure the resource. Hence this algorithm incorporates resource allocation along with scheduling. The task to resource mapping is defined within the same algorithm.

6. Benchmarking scenario

6.1. Task graphs

The control data flow graph based representation of the application is necessary to avoid the architecture dependant variation in the system. Due to the absence of well-defined benchmarks in co-design, randomly generated task graphs were used to

![Fig. 4. Task graph with 200 nodes.](image-url)
represent the sequence of events in embedded system application. Such an approach was necessary in order to avoid architecture dependent variations in the representation of a problem. These task graphs allow representation of any real world application by making parametric changes to the task graph generation process (see Section 7). These task graphs were generated using TGFF, described in [1]. They provide the sequence of operations in a system. This in turn provides the precedence relationship between tasks, which is not necessarily domain specific. A sample task graph is illustrated in Fig. 4. A set of five task graphs with a node count of 200 (on an average, with a variance of one) was generated. The types of nodes were restricted to two, based on the number of inputs/outputs to each node. The variation in the number of input/outputs to the nodes is mapped as a variation in the task functionality. Attributes such as execution time, cost, area and power may also be added to each of the tasks to be randomly assigned. System resources have predefined execution time and power rating for resources. Individual task deadlines are ignored. To accommodate two types of tasks, two types of resources are required. Consequently, the resources differ in the execution time and power consumed. Reconfiguration allows change in functionality between the two types of tasks, with the additional overhead of reconfiguration time. Following is the list of parameters used to generate the task graphs for our experiments.

### 7. Parameters to generate TGFF

Following is the input file for the TGFF generator.

```
tg_cnt 5
task_cnt 200 1
prob_multi_start_nodes 1
task_degree 2 2
period_mul 1, 1, 1, 1, 1
tg_write
eps_write
cvg_write$table_label COMMUN$table_cnt 3$table_attrib price 80 20$type_attrib exec_time 50 20
trans_write
```

The following assumptions are made, only for illustrative purposes, in order to logically evaluate the algorithm used. These parameters may be changed according to the actual values in a real application. The comparative relations between these delays were assumed in accordance with [14] and resemble most real world applications. They are expressed as follows:

\[
t_s = 10 * t_h (16)
\]
\[
t_h = t_e (17)
\]
\[
t_{sh} = t_{hs} (18)
\]
\[
t_{hh} < t_{sh} (19)
\]
\[
t_{hh} = t_{ss} (20)
\]

Similarly for power,

\[
P_s < P_h (21)
\]

The additional power consumed can be expressed as follows:

\[
P_{sh} = P_{hs} (22)
\]
\[
P_{hh} > P_{ss} (23)
\]

#### 7.1. GA parameters

After performing a series of experiments the GA parameters were fixed. A population size of 4000

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unit Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{sw}$</td>
<td>30</td>
</tr>
<tr>
<td>$T_{hw1}$</td>
<td>3</td>
</tr>
<tr>
<td>$T_{hw2}$</td>
<td>4</td>
</tr>
<tr>
<td>$T_{hh}$</td>
<td>1</td>
</tr>
<tr>
<td>$T_{hs}$</td>
<td>2</td>
</tr>
<tr>
<td>$T_{ss}$</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Unit Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_{ss}$</td>
<td>1</td>
</tr>
<tr>
<td>$P_{sh}$</td>
<td>2</td>
</tr>
<tr>
<td>$P_{hs}$</td>
<td>2</td>
</tr>
<tr>
<td>$P_{sh}$</td>
<td>3</td>
</tr>
<tr>
<td>$P_{hh}$</td>
<td>2</td>
</tr>
<tr>
<td>$P_{h1}$</td>
<td>3</td>
</tr>
<tr>
<td>$P_{h2}$</td>
<td>3</td>
</tr>
<tr>
<td>$P_s$</td>
<td>1</td>
</tr>
<tr>
<td>$P_r$</td>
<td>6</td>
</tr>
</tbody>
</table>
was set for all the experiments. The initial population was created randomly. GA was set to evolve over 200 generations. The reproduction parameters were set as $P_c = 0.9$ and $P_m = 0.005$. The time and power parameters assumed for the experiments are listed in Tables 2 and 3.

8. System parameters

9. Experiments

9.1. Experiment 1

To validate the assumptions and to confirm the consistency, the proposed algorithm was executed multiple times, with each run starting with a randomly generated seed. Further, a comparative study was conducted to observe the variations of the results with the introduction of runtime reconfiguration.

9.1.1. Case 1

The system under consideration comprises of two types of resources, they are software and hardware. The hardware resources are assumed to be fixed and pre-configured. To accommodate the two functionally diverse nodes, the FPGA has two resources of each type, pre-configured. An equal number of software resources are assumed. Thus, for this configuration the architecture comprises of one FPGA with four resources and four GPPs.

9.1.2. Case 2

Run time reconfiguration of hardware resource is introduced in this architecture. Consequently, to maintain the same ratio of resources, there are four hardware resources available on the FPGA. Two of which are resources with fixed configurations (pre-configured) and another two can be reconfigured at runtime. The same numbers of software resources are retained, as in Case 1. This system architecture comprises of an FPGA and four GPPs, as in Case 1. Case 1 and Case 2 are compared for the design objectives of time, power and simultaneous optimization of time and power.

9.2. Experiment 2

Along with the design objective of optimization in time and power, GA is used to determine the optimal number of resources required for the implementation of task graph. Three kinds of resources are assumed to exist concurrently in the system architecture, namely, software resources, fixed pre-configured hardware and runtime reconfigurable hardware resources.

9.2.1. Case 1 – Time constraint

A hard-deadline is set, to introduce a time constraint. The penalty function is applicable whenever there is a time violation. This was experimented for optimization in power and the multi-objective optimization of time and power. To determine the hard deadline, the following two parameters were introduced. Firstly, $H_{\text{Critical}}$ was set on the basis of the calculation of the ASAPTime, with only hardware resources. On the basis of this value, the algorithm was executed with a hard-deadline of $1.5 * H_{\text{Critical}}$ and a deadline of $2 * H_{\text{Critical}}$.

9.2.2. Case 2 – Power constraint

A restriction in power is introduced, with a power constraint. Similar to the hard-deadline in Case 1, a penalty function was introduced to limit the maximum power requirements. This case of power constraining was experimented for the optimization objective of time and the multi-objective optimization of time and power. The maximum power limitation was determined based on the maximum power consumed by the system, defined by $P_{\text{max}}$ and $S_{\text{max}}$ in Eqs. (7) and (9). The experiments were repeated for a power limitation of $(P_{\text{max}} + S_{\text{max}})/3$, $(P_{\text{max}} + S_{\text{max}})/4$ and $(P_{\text{max}} + S_{\text{max}})/6$.

10. Results

10.1. Comparison of Random search with GA

The plot in Fig. 5 compares the execution times and power consumed for varying values of $P_c$ and $P_m$. The values to the extreme left, ($P_c = 0$ and $P_m = 1$), are sets of random numbers generated. The value to the extreme right, ($P_c = 0.9$ and $P_m = 0.005$), are the sets used in the GA experiments.
\( P_m = 1 \) represent a case of Random search. This, when compared to the values appearing at the center of the graph (\( P_c = 0.9 \) and \( P_m = 0.01 \)) and towards the extreme right (\( P_c = 0.9 \) and \( P_m = 0.005 \)), show a very large variation. This illustrates the significance of the optimization procedure adapted. The plot of the fitness function for the maximization objective determined for one of the experiments is shown in Fig. 6. The fitness function stabilizes towards the end of the pre-defined number of generations, indicating optimal results.

10.2. Results from Experiment 1

Tables 4–6 summarize the performance for the optimization in time and power individually and the combined optimization of the two objectives. These results are for a resource constraint described in Case 1. Tables 7–9 show the performance variation with the introduction of run-time reconfiguration on hardware. The resource constraints are as mentioned in Case 2. Following sections describe the inferences made for the results obtained.

### 10.2.1. Consistency of GA

The results represented in Tables 4 (and 5, 6) and 7 (and 8, 9) show that GA arrives at the similar set of results for all the 10 random executions of the algorithm. This validates the fitness function employed and demonstrates that the algorithm employed is consistent at arriving at the optimal values.
10.2.2. Time optimization

For time critical applications it is seen that run-time reconfiguration enhances system performance. This improvement is seen even with the presence of the additional reconfiguration overhead. The results for timing optimization so obtained with reconfigurable hardware resources are better than the results with pre-configured hardware. Fig. 7 shows the variation in time in the two cases, for all the design objectives.

10.2.3. Power optimization

For power optimization, the procedure almost completely avoids hardware blocks to reduce the power consumed by the system in both the cases. The task execution relies entirely on the software resource and all the operations are queued to this resource. Hence, for power critical applications, the choice of reconfiguration may be avoided, since its introduction only deteriorates power. Fig. 8 compares the results obtained for power in the two cases, for all the design objectives.

10.2.4. Time and Power optimization

For the multi-objective optimization in power and time to reduce the total time and power consumed simultaneously, both hardware and software design units are utilized. From Figs. 7 and 8, it is observed that, introduction of time as parameter along with power changes achieves results similar to the individual optimization of time.

10.2.5. Task distribution

Fig. 9 depicts the variation in the resource utilization in hardware and software elements, depending on the objective applied. As can be observed, an all-hardware implementation is best suited for time optimization. The software units utilized in this process merely provide an additional resource for parallel operation. Similarly, for power optimization, the procedure almost completely avoids hardware blocks to reduce the power consumed by the system. The task execution relies entirely on the software resource and all the operations are queued to this resource. The result so obtained is the best combination for a power-optimal solution, but unacceptable for time critical applications. For the multi-objective optimization in power and time, the influence of inter-processor communication is insignificant. To reduce the total time and power consumed both hardware and software design units are utilized. This results in a solution better than pure-software or pure-hardware implementation.

A similar observation may be made for the reconfigurable hardware architecture, shown in Fig. 10. Among the hardware resources utilized, the reconfigurable resources outnumber pre-configured hardware.
10.3. Results from Experiment 2

10.3.1. Results with Hard deadline

Table 10 summarizes the results obtained for the optimization of power, while meeting a hard-deadline. Table 11 lists the results obtained for the optimization of time and power for a set hard-deadline. Limiting the execution time, implements a power efficient design well within the specified deadline.

![Task distribution of tasks on hardware/software – Case 2.](image)

This is implemented with the minimum number of resources.

10.3.2. Results with Power limitation

Table 12 summarizes the results obtained for the optimization of time, while meeting the maximum power constraint. Table 13 lists the results obtained for the optimization of time and power for a set maximum power requirement. The variation in resource allocation and task mapping causes a variation in time and power. The penalty introduced to restrict the power consumption, limits the search space. Hence, this results in timing optimization while meeting the maximum power requirement. From these results it is seen that, as the number of tasks mapped on a resource type increases, the resource count is proportionally kept high. Similarly for low resource utilization, the resource count is kept to the minimum required. This indicates usage of software elements in time critical applications only for tasks which do not affect the total execution time. Similarly, resource count on hardware is restricted to the minimum possible for power

### Table 10
Power optimization with hard deadline

<table>
<thead>
<tr>
<th>Objective</th>
<th>AsapTime on hardware</th>
<th>AsapTime on hardware * 1.5</th>
<th>AsapTime on hardware * 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deadline</td>
<td>260</td>
<td>390</td>
<td>520</td>
</tr>
<tr>
<td>Time</td>
<td>295</td>
<td>390</td>
<td>515</td>
</tr>
<tr>
<td>Power</td>
<td><strong>1183</strong></td>
<td><strong>1105</strong></td>
<td><strong>1031</strong></td>
</tr>
<tr>
<td>SW Resources</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Reconfigurable HW Resources</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>HW Resources of Type1</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>HW Resources of Type2</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Tasks on SW</td>
<td>8</td>
<td>18</td>
<td>32</td>
</tr>
<tr>
<td>Tasks on Reconfigurable HW</td>
<td>60</td>
<td>47</td>
<td>46</td>
</tr>
<tr>
<td>Tasks on HW</td>
<td>134</td>
<td>137</td>
<td>124</td>
</tr>
</tbody>
</table>

### Table 11
Time and Power optimization with hard deadline

<table>
<thead>
<tr>
<th>Objective</th>
<th>AsapTime on hardware</th>
<th>AsapTime on hardware * 1.5</th>
<th>AsapTime on hardware * 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deadline</td>
<td>260</td>
<td>390</td>
<td>520</td>
</tr>
<tr>
<td>Time</td>
<td><strong>294</strong></td>
<td><strong>319</strong></td>
<td><strong>449</strong></td>
</tr>
<tr>
<td>Power</td>
<td><strong>1194</strong></td>
<td><strong>1145</strong></td>
<td><strong>1082</strong></td>
</tr>
<tr>
<td>SW Resources</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Reconfigurable HW Resources</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>HW Resources of Type1</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>HW Resources of Type2</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Tasks on SW</td>
<td>6</td>
<td>12</td>
<td>27</td>
</tr>
<tr>
<td>Tasks on Reconfigurable HW</td>
<td>60</td>
<td>43</td>
<td>41</td>
</tr>
<tr>
<td>Tasks on HW</td>
<td>136</td>
<td>147</td>
<td>134</td>
</tr>
</tbody>
</table>
optimal solutions. The multi-objective optimization of power and time arrives at a distributed resource allocation.

11. Conclusions

A consistency in the quality of the results obtained using GA is observed over a wide range of design examples. The analysis in terms of the mean and variation ranges establishes the dependability of GA in the problem addressed. Retaining the same scheduling algorithm, the fitness function can be modified to redesign the system objective. Thus, the proposed optimization technique for time and power can be extended to other design parameters such as area and cost of silicon. The suitability of using objective-specific resources is clearly observed in the resource utilization ratios. Resource allocation using GA provides information to build the most suited target architecture for a system application. With this approach, the number of unused resources and under-utilized devices are reduced to a minimum possible. Hence, for a given application, with set design objectives, GAs arrive at the most suited resource allocation and scheduling for the system. In short, GA evolves the design of the system. Additionally, with the introduction of penalties for time and power, the power optimization keeps a limit on the execution time. Also, the time optimization maintains power below the maximum threshold. On account of the consistency in the results obtained, GA can be used as a tool for future design enhancements in hardware–software co-design. A nested GA can be formulated on the hardware resources to arrive at the best possible combination of reconfigurable hardware resources. A similar second level of GA can be introduced to decide the resource-level granularity. Broadly, GA as a partitioning tool may be used in future to design adaptive system architectures by changing the task-resource mapping. To introduce weights for the objective functions with a greater priority, the fitness function may be transformed as follows:

\[ \text{Fitness} = f_{\text{time}} \times f_{\text{power}} \]  \hspace{1cm} (24)

\[ \text{Fitness} = f_{\text{time}}^\alpha \times f_{\text{power}}^\beta \]  \hspace{1cm} (25)

| Table 12 | Time optimization with Power limitation |
|---|---|---|
| \((P_{\text{max}} + S_{\text{max}})/6\) | \((P_{\text{max}} + S_{\text{max}})/4\) | \((P_{\text{max}} + S_{\text{max}})/3\) |
| MaxPower | 400 | 600 | 801 |
| Power | 467 | 600 | 793 |
| Time | **1678** | **1404** | **1004** |
| SW Resources | 4 | 4 | 4 |
| Reconfigurable Resources | 1 | 1 | 1 |
| HW Resources of Type1 | 1 | 1 | 3 |
| HW Resources of Type2 | 1 | 1 | 1 |
| Tasks on SW | 194 | 156 | 99 |
| Tasks on Reconfigurable HW | 2 | 16 | 29 |
| Tasks on HW | 6 | 30 | 74 |

| Table 13 | Time and Power optimization with Power limitation |
|---|---|---|
| \((P_{\text{max}} + S_{\text{max}})/6\) | \((P_{\text{max}} + S_{\text{max}})/4\) | \((P_{\text{max}} + S_{\text{max}})/3\) |
| MaxPower | 400 | 600 | 801 |
| Power | **457** | **503** | **537** |
| Time | **1703** | **1577** | **1523** |
| SW Resources | 4 | 4 | 4 |
| Reconfigurable HW Resources | 1 | 1 | 1 |
| HW Resources of Type1 | 1 | 2 | 1 |
| HW Resources of Type2 | 1 | 1 | 1 |
| Tasks on SW | 199 | 179 | 172 |
| Tasks on Reconfigurable HW | 1 | 4 | 6 |
| Tasks on HW | 2 | 19 | 24 |
where \( a \) and \( b \) represent the weights which prioritize the objective. When \( a \) is \( \geq 1 \), the value of \( f_{\text{time}} \) has a greater significance than with \( a < 1 \).

The approach proposed here is generic and can be applied to any real world problem. The aim of conducting these experiments was to put forth the concept of task graph based application representation and the application of GA for the partitioning. This could be further extended to integration into a design automation environment to study the suitability of our approach, however, it is presently beyond the scope of this paper.

Acknowledgments

We thank the reviewers for their valuable comments. The support of Natural Sciences and Engineering Research Council of Canada (NSERC) and Alberta Ingenuity is gratefully acknowledged.

References