Towards Dilated Placement of Dynamic NoC Cores

Branislav Hredzak[1], Oliver Diessel[2]

[1] School of Electrical Engineering
[2] School of Computer Science and Engineering
University of New South Wales
Sydney NSW 2052, Australia
b.hredzak@unsw.edu.au, odiessel@cse.unsw.edu.au

Abstract. Instead of mapping application task graphs in a compact manner onto reconfigurable devices using a network-on-chip for interconnecting application cores, we propose dilating the mappings as much as the available latencies on critical connections allow. In a dilated mapping, the unused resources between an application's configured components can be used to provide additional flexibility when the configuration needs to change. We motivate the reasons for dilating application task graphs targeted at reconfigurable devices; derive a simulated annealing approach to dilating the placement of such graphs; and present preliminary results of applying the algorithm to synthetic test cases. The method appears to result in successful and meaningful graph dilation and could be further tuned to satisfy desired power constraints.

Keywords. Modular reconfiguration, networks-on-chip, application mapping, dilation

1 Introduction

Our research aims to facilitate the implementation, using field-programmable gate arrays, of performance critical digital systems in which hardware components need to rapidly adapt to changing requirements. Relevant systems include applications in video surveillance, mobile communications, multimedia, telemedicine, robotics, remote environments including space, and electronic warfare. It is our belief that such dynamically reconfigurable systems are not adequately supported with frameworks for their development. This leads to costly ad-hoc development efforts, design failures, re-invention, and a lack of generality that disables re-use. We aim to discover new, more general approaches to improving the utilization and flexibility of FPGAs when used for dynamic reconfiguration.

Central to our approach is our premise that dynamically reconfigured hardware cores should be interconnected by a network-on-chip that can be adapted to optimize the communication between components. In contrast to conventional

Dagstuhl Seminar Proceedings 10281
Dynamically Reconfigurable Architectures
http://drops.dagstuhl.de/opus/volltexte/2010/2834

approaches, which assume FPGA resources are in short supply, we propose exploring the potential of utilizing the considerable resources that are expected to be available in the mid-term future. Instead of mapping an application to the smallest compact region possible, we propose the simple yet unconventional approach of *dilating* the mapping as much as the available latency (slack) on critical network connections allows.

In a dilated mapping, the free space comprising unused configurable regions, associated network routers and links between an application's configured components can be used to provide additional flexibility when the configuration is required to be changed. For example, when a core is to be added to a dilated design, there are potentially many more locations to find placements that provide good connectivity to components which have already been configured. Alternatively, it will be less disruptive, and is likely to incur less overhead, to move within a small neighbourhood (jog) some of the cores that have already been placed in order to make room for a new one. More importantly, it will be easier to allocate additional routing paths or insert express channels into free regions when additional bandwidth is required or latency needs to be reduced.

Dilating the placement will cause more power to be consumed due to the additional network links and routers used. However, this increase in consumption is relatively easy to estimate, and if the power consumed by a dilated configuration is deemed too high, we intend *compacting* the arrangement of cores to reduce path lengths and to switch off those regions that have thereby been rendered idle.

This idea requires us to investigate several problems, including (1) techniques for dilating the placement of application task graphs, (2) methods for modifying a dilated placement of cores through addition of new cores, removal of existing cores, and updates to inter-core connections such as changes in bandwidth and latency, (3) approaches to compacting a dilated placement in order to reduce power consumption or latency, and (4) assessment of the techniques on benchmark applications to determine their efficacy in terms of performance and energy use.

The principal contributions of this paper are: (1) an outline of the potential benefits of the concept of dilating the mapping of dynamic task graphs, (2) the derivation of a simulated annealing algorithm for dilating the placement of an application task graph, and (3) an analysis of the preliminary results we obtained by applying the algorithm to a couple of simple test cases.

The paper is organized as follows. In the next section we detail the background that motivated us to examine the potential benefits of dilating the mapping of application task graphs into network-on-chip based FPGAs. In Section 3 we derive the objective function of a simulated annealing algorithm for dilating the mapping of a task graph. We describe our simulation experiments and results in Section 4 and conclude in Section 5 with a summary of the objectives and results of our work so far and outline the directions for further study.

2 Background

Reconfigurable logic allows application components to be accelerated over their execution as software [1]. Reconfigurability also provides a means by which performance-critical components can be implemented in a more flexible, robust, reusable and useful manner than if they are provided as ASIC devices. Reconfigurable systems provide designers with the flexibility to accommodate various system changes. The 2009 semiconductor industry roadmap estimates that approximately 35% of current system on chip functionality implemented in software or hardware is reconfigurable. By 2024, this fraction is expected to rise to a massive 70% [2]. Combined with the scaling effect due to Moore's Law (approximately 1000-fold over the next 15 years), powerful new design techniques will be needed to exploit this potential.

Having been configured for a particular application, systems that include reconfigurable logic can be *reconfigured*, meaning new functional blocks or components can be added and/or old ones can be removed or exchanged, for a range of benefits. For example, changes may be initiated by a user who requires new functionality to be added, such as a specific filter to a surveillance camera network [3]. Alternatively, the user may want to improve overall system performance by choosing a more efficient implementation of some component. When the power supply to the system is disconnected the system operating objective may need to switch from a high-performance mode to a low power mode [4]. The system environment can change and the system needs to cope with these. For example, a signal may become affected by noise and more filtering is then required to process the signal [5]. A different modulation scheme may need to be employed [6], or some (hardware) component may have failed, necessitating reallocation of functional tasks to the available resources [7]. Furthermore, the system designer may exploit the fact that certain components are only needed sometimes, or that others are never needed simultaneously in order to conserve the amount of logic or power used at runtime [8]. As suggested, some of these changes may be planned for at design time, but it is desirable to allow for others to occur in an unforeseen manner as repairs to bugs become available, when a new component is developed, or when new protocols are invoked, or when a user changes the requirements or demands new functionality.

When the extent of possible changes is *closed* or planned for, the designer can optimize the system architecture and operation a priori. This is the way embedded systems are commonly designed. However, it may be desirable to have mechanisms in place that allow a system to cope with *open* or unforeseen application changes and to cope with them dynamically, while the system is in operation, instead of after a potentially lengthy offline re-optimization and restart. It is also worth noting that a system may not be concerned with performing just one task or serving a single application. It may be dealing with many complex or compute-intensive tasks simultaneously [9,10]. And in such a case, the changes in one application may impact on the other applications that are sharing the resources of the system and calls for a response in executing applications to dynamic changes in resource availability and system load [11].

Engineering digital systems that can dynamically respond to changes in the number, type and interconnection of components they are composed of, even when known up front, is non-trivial. The following significant issues need to be resolved to obtain a generally applicable solution, and apply whether the envisaged system environment is closed or open, namely: (1) identifying and designing the components involved in dynamic reconfiguration; (2) designing a reconfigurable system architecture that lends itself to dynamically swapping functional components and their interconnection patterns; (3) controlling the reconfiguration process and maintaining system correctness during the reconfiguration process; and (4) ensuring the envisaged benefits are not swamped by the costs.

The first issue requires methods for determining which components from a set of executing applications are active over given periods of time. This aids in the effective temporal clustering of logic into a sequence of active configurations. The solution is likely to be non-trivial, since it may rely on accurate forecasts of component activity that are independent of the data being processed. Traces may therefore only be of limited use. In the following we assume a *component* or a *core* is a functional block such as an FFT or DCT or matrix multiply unit of the required size. A component may be available as a (hard) IP core or a macro for which a (relocatable) mapping to FPGA resources is already available.

The second issue deals with providing a system structure that can support the dynamic allocation of components to resources and can provide the interconnections the currently active components require. Primarily, this issue is concerned with providing a flexible and efficient method for dealing with time-varying interconnection patterns that include changes to the number and location of interconnected components and changes to the traffic patterns between them.

The third issue is concerned with scheduling and controlling reconfiguration events such that system correctness is never compromised. This includes rerouting traffic, buffering data, disconnecting and reconnecting components to the communication infrastructure, moving components amongst the available resources to ensure constraints can be met, determining a new arrangement and utilization of components such that the objectives of each active application and the system overall can be met.

The fourth issue is concerned with making the effort worthwhile. While some researchers are studying the tradeoffs between execution time or energy and the choice of reconfigurable components, current state of the art relies primarily on principles such as minimizing reconfiguration overheads and maximizing the ratio of the periods during which components are used and the periods during which they are being reconfigured.

In our research, we assume we are provided with an application communication graph which provides a partitioning of an application into configurable components as well as knowledge about the periods during which components are active. We therefore focus on the second issue above and address those aspects of the third that deal with the dynamic interconnect structure and its control. We follow principles that aim to minimize reconfiguration overheads.

2.1 Communication infrastructure

In recent years attention has been drawn towards so-called networks-on-chip [12,13,14,15]. A network-on-chip (NoC) consists of routers that are interconnected via links which convey packetized messages between the routers. A packet contains the data that is to be conveyed from a sending component to a receiving component, which are both connected via network interfaces to routers in the network. The intervening routers make decisions about where a packet is sent next based on the source and destination routers of the packet. In principle, each router provides an opportunity for connecting to components via network interfaces and the network topology generally provides multiple paths to reach any destination. Multiple messages can be conveyed through the network simultaneously. As the number of routers grows, typically the number of connection points and routing paths also grows. Networks are therefore more scalable and flexible than buses and point-to-point connections.

NoC technology appears to be well-suited for providing the communication infrastructure of future dynamically reconfigurable systems-on-chip. However, a number of shortcomings in the previous work can be identified.

First, the previous work concentrates on closed, embedded application sets in which the set of applications and their data communication profiles are known at design time. To date, few results ([16,17,18]) attempt to address the time-varying needs of *dynamic* and *open* application sets. In [16], a regular mesh of routers is proposed. Variously sized cores are placed where they fit and non-deterministic, adaptive routing algorithms are proposed for communicating between the cores, but communication requirements cannot be guaranteed. In [17], a tile-based approach is used to implement a customized network for the configured components. Unfortunately, reconfiguring the tiles disrupts the traffic on links within the tiles. Reconfiguration of the network incurs significant overheads. In order to provide an efficient set of mechanisms for adapting the communication infrastructure to the needs of time-varying applications, the temporal communication requirements of applications must be understood; the degree of run-time flexibility needed should be determined; and the mechanisms by which they are best delivered using reconfigurable logic can be developed.

Second, since they are typically implemented using static resources, NoCs are designed to cater for worst-case communication needs. This means they are over-provisioned and, on average, consume more power than they need to. Implementations in reconfigurable logic make it possible to alter network parameters as execution proceeds. This potential has not yet been systematically studied with due account for the overheads and efficiency of proposed adaptations.

Third, FPGAs are currently quite constrained in the types of specialized resources available and their distribution. While significant further specialization can be expected over the coming decade, and theses shifts in the technology need to be factored into the conclusions of research studies, current resource constraints require NoCs for FPGAs to be designed with consideration for the on-chip complexity of the strategies employed to support dynamic adaptation. The balance between distributed and centralized, autonomous and commanddriven, on-chip and off-chip control measures deserves closer examination than it has so far received.

Fourth, reusable methods for supporting dynamic reconfiguration need to be made available to designers. This goal may require the support of FPGA vendors who vigorously protect the details of the implementation of their devices. Researchers can do little more than prove their methods and suggest how their results could be integrated with vendor tool-flows.

In this paper, we report on the development of a dilated mapping technique that will enable initial, quasi-optimal, dilated placement of application task graphs into a given network topology. The developed mapping technique addresses the following critical issues: (1) how to spread out the placement of cores without exceeding latency requirements; (2) how to place cores so as to maximize the potential to provide additional bandwidth or reduce latency between cores when required; and (3) how to place cores so as to allow fast and easy relocation, addition or removal of cores. Our results also aim to assess the increases in latency and power consumption that result when a task graph is dilated when compared with the more compact mappings conventional placement approaches produce. It is necessary that these objectives be met before we begin to study methods for dynamically updating the cores and the routes used to interconnect them.

3 Development of dilated mappuing techniques

3.1 Definitions

Definition 1: We define a *communication task graph* as a special case of an application characterization graph, as given in [19]. A communication task graph *CTG = CTG(T, C)* is a directed graph, where each vertex $t_i \in T$ represents an IP core or a task that has been mapped to a core and each directed arc $c_{i,j} \in C$ summarizes the total communication or interconnection between vertices t_i and t_j.

While cores for reconfigurable computing applications should be tagged with application-specific information such as size or area, clock frequency, period of activity, and latency, we ignore these details in this paper. However, our work allows the placement of a core to be locked to a particular position to use location-dependent resources.

Each $c_{i,j}$ is tagged with application-specific information including the sum of the bandwidth required by all signals from t_i to t_j and the maximum latency that can be tolerated for the connection.

A communication task graph may be specified by the number of cores in the graph and a listing of the connection characteristics as provided in Table 1 and illustrated in Fig. 1.

Definition 2: We define a *network topology* as a directed graph *NT = NT(R, L)* comprising a set of *routers R* and *links L* connecting them. Each $r_i \in R$ is specified by its *xy* coordinates on an FPGA floorplan and each $l_{i,j} \in L$ connects a given source router r_i with a unique destination router r_j. In this study, each link $l_{i,j}$ has a maximum bandwidth capacity and a latency associated with it. In this work all links are assumed to have the same maximum bandwidth and the same, constant link latency. The routers are assumed to be capable of switching the traffic when all incoming links are operating at maximum bandwidth. In this work, we assume the latency of a router is a constant, irrespective of the traffic volume it switches. Routers are also assumed to have a pair of special *local links* that allow a single core to be directly connected with the router. This pair of links allows the core to inject traffic (data) into the network and to receive data from other cores connected to the network. We define the latency of a network *hop* to comprise the sum of the latency of a single network link and the latency of the router the link inputs to. We neglect the latency, incurred at a core, of injecting data into the network and transferring it from the router to the core. The Manhattan distance between cores and the latency per hop thus characterize the latency between cores.

Table 1. A simple 4-core CTG example.

Connection	**Bandwidth (MB/s)**	**Latency (ns)**
$c_{1,2}$	20	10
$c_{2,3}$	30	20
$c_{3,4}$	40	20
$c_{4,1}$	10	10

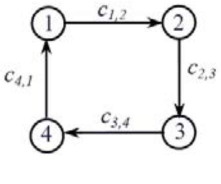

Fig. 1. Example of a simple 4-core CTG.

In this work, we consider the problem of mapping given CTGs to two-dimensional (2D) mesh network topologies. A 2D mesh network has routers located at all *xy* positions in [1..*max_x*, 1..*max_y*] and all routers are connected via bi-directional links to their immediate neighbours to the north, south, east and west, where these neighbours are present.

Definition 3: Mapping a CTG to a network topology involves finding an optimal placement (router positions) for the CTG cores and determining legal, minimal cost routes through network links for the inter-core connections. In our work to date we have only considered XY routing, which is known to be deadlock-free on a mesh [20].

Optimality of the placement can be determined with respect to a number of metrics. Typical *performance* metrics include average/maximum packet latency, bisection bandwidth, network throughput and quality of service, whereas *cost* metrics include average/peak energy/power consumption, network area overhead, total area and average/peak temperature [19]. Maximizing performance usually forces the mapping to be as compact as possible. For example, the CTG of Table 1 might be mapped to a 4 x 4 router mesh as in Fig. 2. In this case, the maximum link bandwidth must be at least 40MB/s for the mapping to be possible with XY routing.

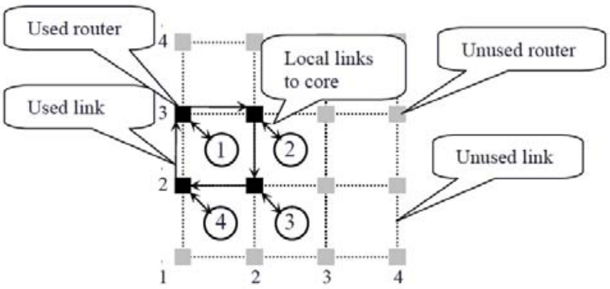

Fig. 2. Mapping of CTG from Table 1 to a 4 x 4 mesh. Router positions are depicted as grey squares at link intersections. Link positions are dotted. Used routers and links are darkened. Positions are numbered along x (horizontal) and y (vertical) axes.

In our work, we are primarily interested in obtaining mappings that facilitate future dynamic modification. Our strategy for achieving such mappings is to dilate the embedding of the cores in the network topology so as to maximize the opportunities (via unused neighbouring routers) for inserting cores into the CTG. The dilation of the mapping is constrained by the latency bounds on connections and the utilization of link bandwidth capacity. Dilating the mapping results in an increase in power that can readily be used to constrain the placement as well. As an example, the CTG of Table 1 can be mapped in a dilated manner to a 4 x 4 router mesh with a maximum link bandwidth of at least 40MB/s and a hop latency of 10ns as in Fig. 3. The difference with the previous mapping of Fig. 2 is that connections $c_{2,3}$ and $c_{3,4}$ have now been mapped to routes that require two hops instead of one as permitted by their given latency constraints.

The advantage of doing so is that if a 5^{th} core needs to be added to the CTG of Table 1, as listed in Table 2, then the mapping of Fig. 2 will need to be modified to accommodate the core with the possibility of incurring significant reconfiguration overheads. However, in this case, the new core can easily be

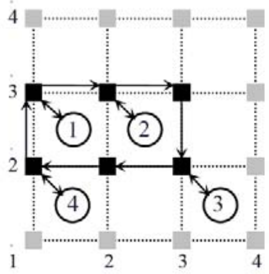

Fig. 3. Dilated mapping of CTG from Table 1 to a 4 x 4 mesh.

inserted into the dilated mapping of Fig. 3 at router position (2, 2) as illustrated in Fig. 4.

Table 2. A 5th core is added to the CTG example of Table 1.

Connection	**Bandwidth (MB/s)**	**Latency (ns)**
$c_{1,2}$	20	10
$c_{2,3}$	30	20
$c_{3,4}$	40	20
$c_{4,1}$	10	10
$c_{4,5}$	10	10
$c_{5,2}$	20	10

The dilated mapping problem we have outlined above in some sense inverts the conventional mapping problem which aims to minimize the mapping cost by minimizing the sum of the products of the embedded connection lengths and the required connection bandwidths. In our case the connection lengths are to be maximized subject to the latency constraints and/or a power constraint. We conjecture this problem is harder to optimize than the usual mapping problem since there is more freedom to place the cores at a distance. As the conventional mapping problem is known to be a special case of the quadratic assignment problem, and therefore NP-hard, we suspect this problem is in NP as well. The approach we have therefore taken to obtain a timely solution to the problem is to apply a simulated annealing heuristic, which is known to produce good results for the conventional mapping problem.

3.2 Simulated annealing framework

Simulated annealing [21], has been applied to combinatorial optimization problems including partitioning, placement and routing, to name a few. The method

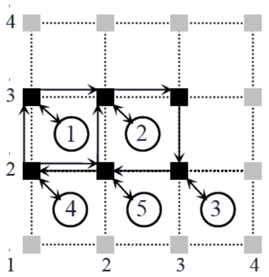

Fig. 4. Addition of 5th core to mapping of Fig. 3

requires four ingredients for it to be applicable: a concise description of a configuration of the system; a random generator of rearrangements of the elements in a configuration; a quantitative objective function containing the tradeoffs to be made; and an annealing schedule of the temperatures and length of times for which the system is to be evolved.

For the dilated mapping problem, the mapping of CTG cores to mesh routers represents a system configuration. It should be clear, that the placement of cores at specific routers together with our choice of XY routing algorithm determines which routes are used for the inter-core connections. Rearrangements are generated by randomly choosing two routers, at least one of which should be occupied by a core, and exchanging the cores connected to the chosen routers. The derivation of the quantitative objective function we used will be described in the remainder of this subsection. The annealing schedule we used is described in the following section.

The objective function (Eqn 1) used in our experiments balances a term promoting a compact placement of the communicating cores with another term that stimulates the dilated placement of cores:

$$cost = \alpha * compaction + (1 - \alpha) * dilation \tag{1}$$

The balance of the compaction and dilation terms can be adjusted by modifying the weight given to the multiplicative constant, α. The goal of the annealing process is to minimize the cost as given by (1).

The compaction term of (1) can be expanded as in (2), in which the product of the required bandwidth for each connection between the cores and the number of hops (Manhattan distance) over which the connection is formed is summed over all connections in the graph. This approach is typically used to minimize the distance between the most heavily communicating cores and thus commonly leads to a low power/low latency/high clock frequency solution.

$$compaction = \sum_{connections,c} bandwidth_c * hops_c \tag{2}$$

The dilation term of (1) is expanded as in (3), whereby the amount of slack on connections, the proximity of non-communicating cores and the utilization of link capacity are minimized. These factors can be traded off by adjusting the multiplicative constants, β, γ and δ.

$$\mathit{dilation} = \beta * \mathit{slack} + \gamma * \mathit{proximity} + \delta * \mathit{utilization} \tag{3}$$

The slack of the current placement of the cores is calculated as in (4), where the latency constraint of a connection, the number of hops for the connection and the hop latency determine the slack of the connection. The slack of all connections is used to determine the slack of the placement.

$$\mathit{slack} = \sum_{\mathit{connections},\, c} (\mathit{latency}_c - \mathit{hops}_c * \mathit{latency}_h) \tag{4}$$

Minimizing the slack on all connections does not ensure that tasks which are not connected are maximally separated. To ensure non-communicating components are spread out, we penalize their proximity. Overall cost is reduced by minimizing the proximity term given by (5), in which the Manhattan distance between non-communicating tasks is minimized.

$$\mathit{proximity} = \sum_{\mathit{unconnected\ task\ pairs}\ (u,v)} -\mathit{distance}_{u,v} \tag{5}$$

Placements that minimize the link utilization are preferred in order to provide spare capacity for cores to be added to the communication graph over time. This goal is captured by Eqn 6, in which the used bandwidth of shared network links is minimized. The term comprises a factor that penalizes the number of connections sharing each link and the sum of the bandwidths of the connections sharing the link. Rather than apply Eqn 6 to the CTG embedding in the network, which penalizes longer (dilated) shared paths more than shorter ones, we construct a graph of the embedded CTG connections, thereby inserting pseudo-vertices into the CTG where signal paths meet at shared routers. Eqn 6 is then applied to this graph of the embedding with each shared connection being counted just once.

$$\mathit{utilization} = \sum_{\mathit{shared\ links},\, s} \mathit{connections}_s * \mathit{bandwidth}_s \tag{6}$$

A number of constraints also need to be observed:

$$\mathit{bandwidth}_l - \mathit{bandwidth}_s \geq 0, \forall \mathit{sharedlinks}, s \tag{7}$$

$$\mathit{latency}_c - \mathit{hops}_c * \mathit{latency}_h \geq 0, \forall \mathit{connections}, c \tag{8}$$

Eqn 7 requires that the total bandwidth allocated to a network link must be no greater than the link bandwidth capacity. In particular, the CTG cannot include any connections whose bandwidth requirement exceeds the maximum link bandwidth. Eqn 8 ensures that no connection exceeds its latency constraint.

4 Results

This section presents simulation results for two synthetic test cases. The simulations were performed in Matlab. We first explain the parameter values we chose for the objective function, and then explain the annealing schedule we followed before presenting the problem instances and our results.

As described in the previous section, the balance of the compaction and dilation terms can be adjusted by modifying the weight given to the multiplicative constant α in (1). For simplicity, in the simulation results presented here, the multiplicative constant α was changed in a discrete manner, i.e., initially $\alpha = 1$, which corresponded to compaction only. Once the compaction converged, α was changed to $\alpha = 0$, which corresponded to dilation only. In this way, the initial allocation was first compacted and then dilated.

The advantage of modifying the objective function in a phased manner is that we were able to achieve convergence of the simulated annealing (SA) algorithm more effectively when the initial placements did not adhere to the given constraints. Such illegal initial configurations may occur when legal starting positions are rare and are more likely to occur in large, dense communication task graphs. By running the compaction stage first, the SA algorithm first obtains a compacted solution in which the maximum link bandwidths are unlikely to be exceeded. This intermediate solution also locates the communicating cores as closely as possible, resulting in the maximum possible slack between communicating cores. When the dilation phase of the SA algorithm is started from a compacted solution, the probability of starting the dilation phase with an illegal solution is small and the convergence of the SA algorithm is significantly improved.

Experimentation with a number of synthetic graphs indicated that $\beta = 1$, $\gamma = 0.2$ and $\delta = 0.04$ resulted in a good balance in the contributions to Eqn (3) from the interconnection slack, the proximity of non-communicating cores and the utilization of link capacity. Settings for α, β, γ and δ that result in good performance on a range of benchmarks will be investigated through further study.

In our experiments, the SA was started with an initial (random) placement of the cores and an initial temperature T_{start}, set to 800 for the compaction phase and to 30 for the dilation phase. Following standard SA practice, the current placement was perturbed at the given temperature (as described in Section III.B) and the change in configuration was accepted if the perturbation resulted in a lower cost configuration, or with exponentially decreasing likelihood if it yielded an increase in cost [21]. In both the compaction and dilation phases we decreased the temperature by ΔT when at least 50 successive rejected perturbations preceded a successful one. During the compaction phase, ΔT was set to 0.5, and during the dilation phase it was set to 0.02. In both cases, the algorithm was terminated when $T_{min} = 0$ was reached, or when 10,000 successive perturbations were rejected.

4.1 Case I

In Case I, listed in Table 3, we dilated 8 cores into a 9 x 9 mesh. Fig. 5 shows the initial placement of the 8 cores. Fig. 6 shows the compacted solution and Fig. 7 shows the dilated solution. The maximum allowed link bandwidth was set to 10. The dilated placement has zero slack for all connections, the utilization is minimized, and the Manhattan distances between non-communicating cores were maximized. Comparing placements shown in Fig. 6 and Fig. 7, it can be appreciated that new cores can be inserted more easily into the dilated placement, For example, if a new core needs to be placed close to cores 2, 3, 6 and 7, this can be more readily achieved for the dilated placement without disturbing any existing connections or placements.

Table 3. Interconnection matrix for Case I. Connections listed as *Required Bandwidth/Connection Latency in hops*; cells representing null connections left blank.

Core	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
t_1		10/2			10/4			
t_2	10/2		10/2			10/4		
t_3		10/2		10/2			10/4	
t_4			10/2					10/4
t_5	10/4					10/2		
t_6		10/4			10/2		10/2	
t_7			10/4			10/2		10/2
t_8				10/4			10/2	

At a given operating frequency, the dynamic NoC power consumption is proportional to the utilized bandwidth and the number of network links and routers involved in signalling. Taking the product of the bandwidth per link and the number of links using that bandwidth, we observe the compacted placement of Fig. 6 consumes 200 units of power while the dilated placement of Fig. 7 consumes 560 units. Such analysis ignores the actual contribution of each factor and the power consumed by the cores. Unused cores and links will also consume static power which hasn't been factored in here.

4.2 Case II

In Case II, as listed in Table 4, we dilated 7 cores into an 8 x 8 mesh. Fig. 8 shows the initial (random) placement of the 7 cores. Fig. 9 shows the compacted solution and Fig. 10 shows the dilated solution. The maximum allowed link bandwidth was set to 50 in this case. The dilated placement resulted in a slack of one for two connections ($c_{3,5}$ and $c_{4,7}$), the utilization was minimized, and the Manhattan distances between non-communicating cores were maximized.

To investigate the reasons for the two connections with non-zero slack, we increased β to 10. Nevertheless, we were unable to eliminate connections with

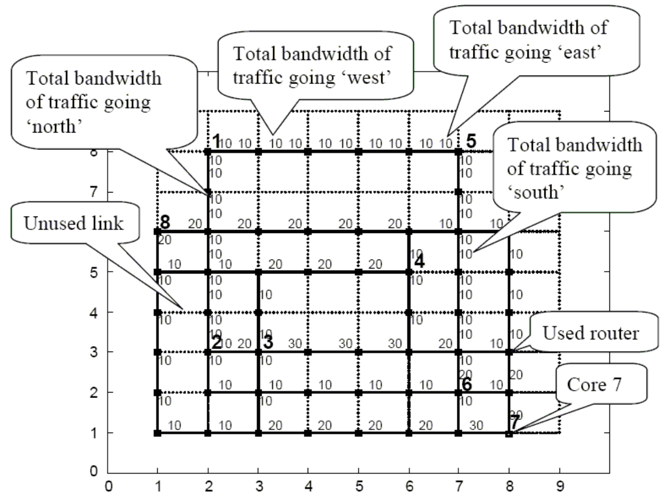
Total bandwidth of traffic going 'north'
Total bandwidth of traffic going 'west'
Total bandwidth of traffic going 'east'
Total bandwidth of traffic going 'south'
Unused link
Used router
Core 7
1
2
3
4
5
6
7
8
0
1
2
3
4
5
6
7
8
9

Fig. 5. Initial (random) placement of the 8 cores into a 9 x 9 mesh.

Table 4. Interconnection matrix for Case II.

Core	t_1	t_2	t_3	t_4	t_5	t_6	t_7
t_1		10/3	11/3		50/3		
t_2	5/3			26/3			
t_3					11/3		
t_4		31/3					23/3
t_5							
t_6		13/3					
t_7			3/3		19/3		

non-zero slack. This suggests that the slack for the two connections may not necessarily represent an optimization error, but that no solution with zero slack for all connections exists.

Applying the same power analysis as for Case I, the NoC used by the compacted placement in this case consumes 236 units of power, whereas the network involved in the dilated placement consumes 522 units in total.

5 Conclusions

Instead of mapping application task graphs to the smallest compact regions possible, we propose *dilating* the mapping as much as the available latency on critical network connections allows. In a dilated mapping, the free space comprising unused configurable regions, associated network routers and links between an application's configured components could be used to provide additional flex-

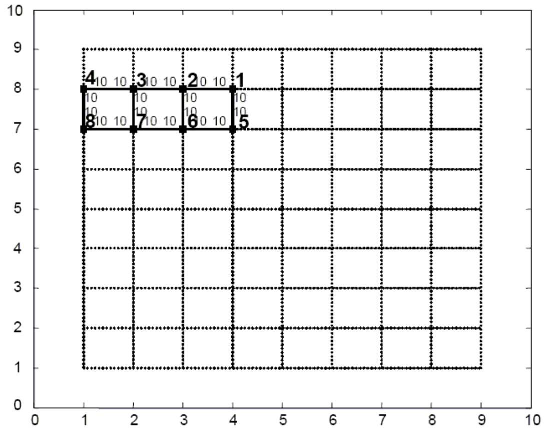

Fig. 6. Placement of 8 cores into a 9 x 9 mesh after compaction.

ibility when the configuration of active cores or their interconnection needs to change.

In this paper we detailed our motivation for exploring the benefits of dilating communication task graphs, derived a simulated annealing (SA) approach to dilating the initial placement of such graphs, and presented preliminary results of applying the algorithm to a pair of synthetic test cases. The method appears to result in successful and meaningful graph dilation and could be further tuned to satisfy desired power constraints since the bandwidth use per link is known at annealing time.

Our future work will involve: (1) performing a systematic study of the SA parameter selection on benchmarks from the literature and synthetic graphs; (2) developing methods for dynamically modifying configurations and benchmarking the methods against doing so without dilation, and (3) developing methods for reducing the power consumption, when desired, by compacting a dilated network.

6 Acknowledgment

We wish to thank Lingkan Gong for his contributions to early discussions on the simulated annealing approach developed in this paper and for his experimental validation of the power assumptions used in our analysis.

References

1. El-Araby, E., Gonzalez, I., El-Ghazawi, T.: Exploiting partial runtime reconfiguration for high-performance reconfigurable computing. ACM Transactions on Reconfigurable Technology and Systems **1** (2009) 23 pp.
2. ITRS: International Technology Roadmap for Semiconductors 2009 Edition — Design. http://public.itrs.net/Links/2009ITRS/Home2009.htm (2009)

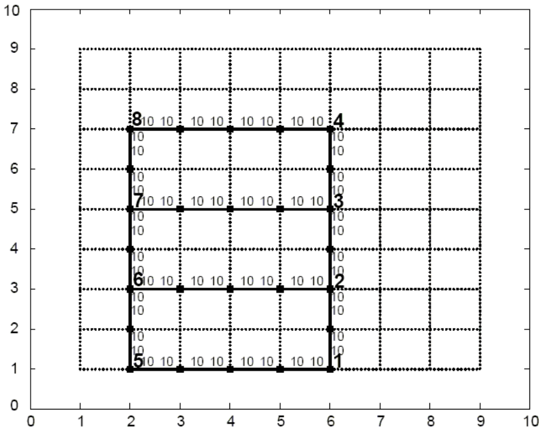

Fig. 7. Placement of 8 cores into a 9 x 9 mesh after dilation.

3. Tadigotla, V., Commuri, S.: Dynamic image filter selection using partially reconfigurable FPGAs for imaging operations. In: International Conference on Circuits, Systems, Electronics, Control and Signal Processing. (2006) 60 – 65
4. Paulsson, K., Hübner, M., Becker, J.: On-line optimization of fpga power-dissipation by exploiting run-time adaption of communication primitives. In: Symposium on Integrated Circuits and Systems Design. (2006) 173 – 178
5. Tessier, R., Swaminathan, S., Ramaswamy, R., Goekel, D., Burleson, W.: A reconfigurable, power-efficient adaptive Viterbi decoder. IEEE Transactions on VLSI Systems **13** (2005) 484 – 488
6. Smit, G., Havinga, P., Smit, L., Heysters, P., Rosien, M.: Dynamic reconfiguration in mobile systems. In: International Conference on Field-Programmable Logic and Applications (FPL). (2002) 171 – 181
7. Zipf, P.: Applying dynamic reconfiguration for fault tolerance in fine-grained logic arrays. IEEE Transactions on VLSI Systems **16** (2008) 134 – 143
8. Burns, J., Donlin, A., Hogg, J., Singh, S., de Wit, M.: A dynamic reconfiguration run-time system. In: IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'97). (1997) 66 – 75
9. Diessel, O., ElGindy, H., Middendorf, M., Schmeck, H., Schmidt, B.: Dynamic scheduling of tasks on partially reconfigurable FPGAs. IEE Proceedings — Computers and Digital Techniques **147** (2000) 181 – 188
10. Simmler, H., Levinson, L., R. Männer, R.: Multitasking on FPGA coprocessors. In: International Workshop on Field-Programmable Logic and Applications (FPL). (2000) 121 – 130
11. Diessel, O., Wigley, G.: Opportunities for operating systems research in reconfigurable computing. Technical report ACRC–99–018, Advanced Computing Research Centre, School of Computer and Information Science, University of South Australia, Mawson Lakes, SA (1999)
12. DeMicheli, G., Benini, L.: Networks on Chips: Technology and Tools. Morgan Kaufmann (2006)
13. Jantsch, A., Tenhunen, H.: Networks-on-Chip. Kluwer (2003)

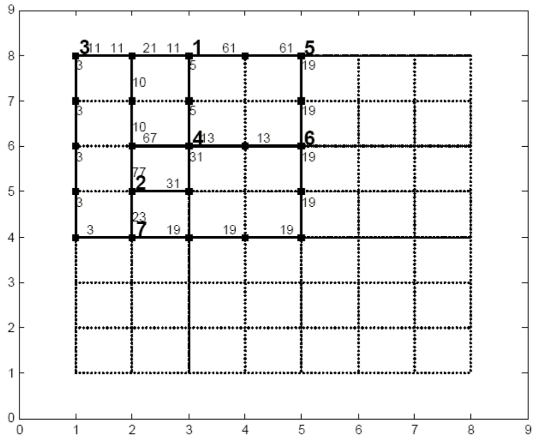

Fig. 8. Initial (random) placement of the 7 cores into a 8 x 8 mesh.

14. Dally, W., Towles, B.: Route packets, not wires: On-chip interconnection networks. In: Design Automation Conference (DAC). (2001) 684 – 689
15. Taylor, M.B., Lee, W., Miller, J., Wentzlaff, D., Bratt, I., Greenwald, B., Hoffmann, H., Johnson, P., Kim, J., Psota, J., Saraf, A., Shnidman, N., Strumpen, V., Frank, M., Amarasinghe, S., Agarwal, A.: Evaluation of the Raw microprocessor: An exposed-wire-delay architecture for ILP and streams. In: International Symposium on Computer Architecture (ISCA). (2004) 2 – 13
16. Bobda, C., Majer, M., Koch, D., Ahmadinia, A., Teich, J.: A dynamic NoC approach for communication in reconfigurable devices. In: International Conference on Field Programmable Logic and Applications (FPL). (2004) 1032 – 1036
17. Pionteck, T., Koch, R., Albrecht, C.: Applying partial reconfiguration to Networks-on-chips. In: International Conference on Field Programmable Logic and Applications (FPL). (2006) 155 – 160
18. Stensgaard, M., Spars, J.: ReNoC: A Network-on-chip architecture with reconfigurable topology. In: International Symposium on Networks-on-Chip (NOCS). (2008) 55 – 64
19. Marculescu, R., Ogras, U., Peh, L.S., Jerger, N., Hoskote, Y.: Outstanding research problems in NoC design: system, microarchitecture, and circuit perspectives. Trans. Comp.-Aided Des. Integ. Cir. Sys. **28** (2009) 3 – 21
20. Ni, L., McKinley, P.: A survey of wormhole routing techniques in direct networks. Computer **26** (1993) 62 – 76
21. Kirkpatrick, S., Gelatt, C. D., J., Vecchi, M.P.: Optimization by simulated annealing. Science **220** (1983) 671 – 680

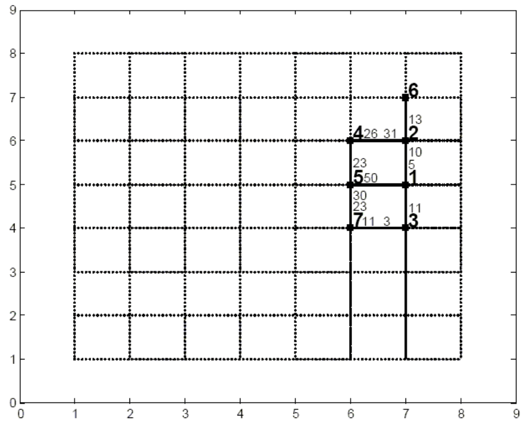

Fig. 9. Placement of 7 cores into a 8 x 8 mesh after compaction.

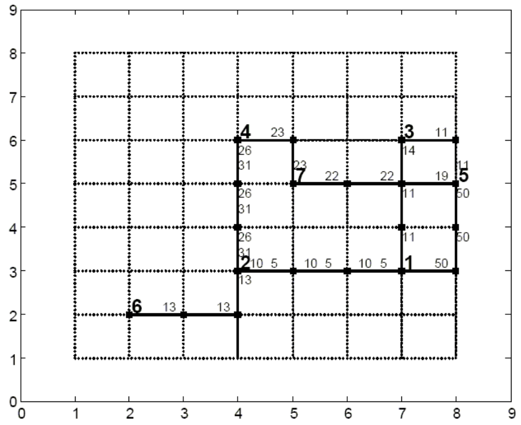

Fig. 10. Placement of 7 cores into a 8 x 8 mesh after dilation.