# Dynamic Architecture Adaptation to Improve Scalability of Sensor Networks: A Case Study for a Smart Sensor for Face Recognition

Yulei Weng, Sankalp Kallakuri, Xiaoyao Liang,
Alex Doboli, Sangjin Hong, Tom Robertazzi, Simona Doboli†
Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY, 11794-2350
† Department of Computer Science, Hofstra University, Hempstead, NY
Email: adoboli@ece.sunysb.edu

## ABSTRACT

*This paper presents a method for dynamic adaptation of sensor node architectures to improve the scalability of a sensor network. The method aids producing control algorithms for on-line adaptation, so that the node's architecture effectively addresses continuously changing operation conditions, like variable data rates and changing latency constraints. To illustrate the method, we discuss a case study for a camera based smart sensor node.*

## 1. INTRODUCTION

Wireless sensor networks are receiving an increasing amount of attention from researchers in universities, government and industry because of their promise to become a revolutionary technology [Estrin,2002; Jones,2001; Kahn,2000]. Besides capabilities specific to any computer network, like connectivity, ubiquity, and trustworthiness, sensor networks require the "intelligence" for scalable, autonomic, and evolvable behavior. Scalability is the quality of a network to optimally address (during execution) the dynamics of operation conditions, including variable input data rates and continuously changing performance requirements. Scalability originates new research problems, as the static concept of optimizing the implementation must be replaced by the more flexible notion of adapting the implementation.

Traditionally, system implementations are optimized for the worst-case operation scenario described by fixed attributes and performance needs. For sensor networks, this design paradigm is quite unsatisfactory, given the scarcity of resources and the difficulty to re-optimize a deployed sensor node for new requirements. Instead, the system must continuously monitor the functioning conditions, and correspondingly react by modifying the sensor node architectures through adding or removing of hardware resources. The algorithm controlling architectural adaptation is embedded either in the middleware of each sensor node, or realized in hardware for having higher speed or lower energy consumption.

This paper proposes a methodology for *dynamic adaptation* of sensor node architectures to improve the scalability of sensor networks. The method aids designing control algorithms for on-line adaptation, so that node architectures effectively address *continuously* changing operation conditions, like input data of modifying sizes or latency constraints of variable lengths. Adaptation is obtained through architectural reconfiguration by adding or removing hardware resources.

We focus on a specific application type, in which the switching between operation modes is accurately modeled by a chain of transitions. We present an original technique for finding reconfiguration conditions. To illustrate the method, we discuss a case study for a camera based sensor node. We show that architecture adaptation is capable of employing the minimal set of resources needed for a task, thus reducing the energy consumption of a node.

The paper is organized in five sections. Section 2 describes the problem, and Section 3 presents the proposed design methodology. Section 4 discusses the case study, and finally conclusions are offered

## 2. PROBLEM DESCRIPTION

Reconfiguration is an important way in developing autonomic, evolvable and scalable behavior to a sensor network. We identified two kinds of reconfiguration, functional reconfiguration and architectural reconfiguration.

- *Functional reconfiguration* implies the modification of a node's connectivity to neighbors or of the algorithms locally executed by a node, like algorithms for data processing, profiling, and identification.
- *Architectural reconfiguration* involves adaptation of the sensor node architecture to changing condition of operation, such as varying data rates and energy levels, new real-time constraints, and so on. Hardware resources are dynamically allocated (through reconfiguration) to the execution threads to meet new performance requirements.

In this paper, we focus on architectural reconfiguration.

Architectural reconfiguration is needed for adapting the architecture of a sensor node to the changing conditions of operation, like variable amounts of input data, decreasing energy levels, modifying latency constraints, different quality of service needs, and so on. For example, for the camera based smart sensor discussed in Section 4, the image processing algorithm might have to process images of different sizes. Also, as the energy level goes down, the processing will be conducted at slower clock frequencies in order to reduce power consumption. The architecture parallelism will have to be increased for meeting the desired latency constraint.

In general, from the application point of view, varying operation conditions cause the change of task execution times, data communication quantity and communication time between tasks, and memory requirements. For example, if input data rates change dynamically then the overall latency for the application has to be also permanently modified. This could happen, if suddenly events of interest occur for the camera based smart sensor node [Brooks,2003]. Photos will have to be taken at higher rates. A naive design solution would dimension the architecture for the worst-case scenario, such as the highest rate at which pictures are taken. This solution will keep many resources unused, which obviously will unnecessarily consume energy. Energy consumption is a significant problem, as the node energy is difficult to replenish [Jones, 2001; Kahn, 2000].

To optimally address the dynamic characteristics of an application, the sensor architecture will have to continuously adapt to the new attributes. Resources will be added or removed from the individual task depending on the task attributes' variations and the criticality of tasks. After usage, hardware resources are allocated to other tasks, so that the amount of unused resources remains low. This is important considering that sensor nodes tend to have scarce amounts of resources (including energy). From design point of view, this type of reconfiguration poses two interesting questions: (1) finding the conditions that trigger resource re-allocation to tasks, and (2) identifying the actual amount of resources assigned to each task.

We have to stress that architectural reconfiguration is quite different from reconfigurable computing [Borgatti,2003]. Reconfigurable computing involves static, off-line synthesis and optimization, whereas architectural reconfiguration involves synthesis for dynamic operating conditions. In reconfigurable computing, the attributes of the application (like task execution times and latency requirements) are static. Synthesis customizes the architecture for the static attributes of the application. In contrast, architectural reconfiguration involves continuous adaptation of the architecture through adding or removing of resources to address the varying conditions of operations. Adaptation is on-line, thus the amount of architectural modification has to be minimal across contiguous conditions of operations.

## 3. DESIGN METHODOLOGY

We suggest an architectural reconfiguration approach, in which each task has allocated a *static set of resources* and a *dynamic set of resources*. In this section, we present the application characteristics and the specifics of the sensor node architecture, and enumerate the steps that compose the design flow for architectural reconfiguration.

During operation, there is a gradual modification of the sensor network behavior (thus of the sensor nodes also), when an event of interest happens. For example, the sensors that detect the vehicle will start collecting images at a continuously higher pace. Once the event disappeared, images will be collected at lower rates. Hence, sensor nodes transit from operation modes for the lower data acquisition rates to the operation modes for higher rates, and then back to those with lower rates. This behavior can be modeled as a chain of transitions.

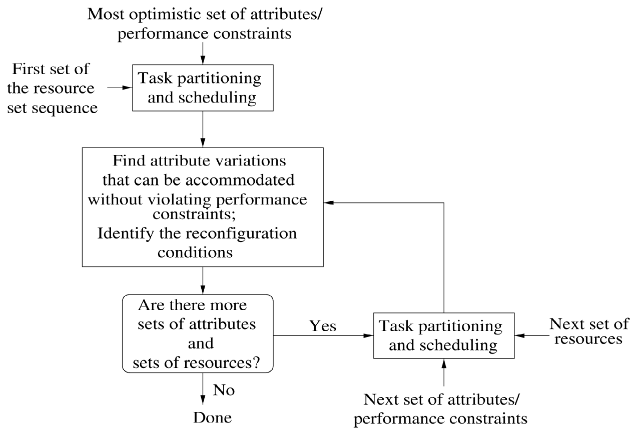
Most optimistic set of attributes/ performance constraints
First set of the resource set sequence
Task partitioning and scheduling
Find attribute variations that can be accommodated without violating performance constraints; Identify the reconfiguration conditions
Are there more sets of attributes and sets of resources?
Yes
No
Done
Task partitioning and scheduling
Next set of resources
Next set of attributes/ performance constraints

*Figure 1*: Design flow for architectural reconfiguration

We consider that the total amount of resources is given. Figure 1 shows the hardware-software co-design flow for architectural reconfiguration. The flow first decides the static set of resources assigned to each task. For this, it uses the most optimistic set of task attributes, such as the shortest execution time for each task or the longest latency constraint. Then, using traditional partitioning and scheduling techniques (e.g., the one in [Doboli,2001]), tasks are mapped and scheduled on the allocated resources.

Next, keeping the resource set unchanged, as well as the mapping and scheduling of tasks and communications, the design flow identifies the maximum amount of attribute variations (in our case execution time variations) that are tolerable without exceeding the required performance, like latency. At the end of this step, reconfiguration conditions are identified that will keep the resource set unmodified as long as these conditions are not violated. Whenever conditions are not met, an event is generated, so that the operation mode of the architecture is modified. Then, for the next set of resources in the sequence, starting from the more critical paths, resources are added (or deleted) to tasks, so that the increased (decreased) task execution times can be accommodated without exceeding the fixed latency. The resources contemplated at this step define the dynamic resource set to be shared across tasks. The algorithm will first tackle the critical paths. This process continues until all operation and performance conditions are fully addressed.

The discussion of the design methodology concludes that, besides traditional partitioning and scheduling algorithms, the synthesis flow ought to include algorithms for calculating the conditions that will trigger modification of the current set of architectural resources, as well as a procedure for finding the dynamic set of resources for each task.

To tackle the problem of identifying the conditions that control architectural reconfigurations, we started by (i) finding closed-formed symbolic expressions for the system performance requirements, followed by (ii) using the symbolic expressions and interval operators in searching for the task execution time intervals that still meet the latency performance needs. If we limit ourselves to conditions that are relations, like $threshold_L \leq execution\ time \leq threshold_R$, then the problem of identifying the reconfiguration condition is equivalent to finding ranges [$threshold_L$, $threshold_R$] for task execution times that still satisfy the fixed latency requirement. This originates an interesting theoretical problem, not tackled - to the best of our knowledge - by the interval arithmetic community [Stolfi,1997]. Typically, interval arithmetic copes with the dual of our problem: finding the tightest range for the result of an expression with intervals as parameters. Interval analysis defines interval operators, like addition, subtraction, multiplication, division, trigonometric functions etc.

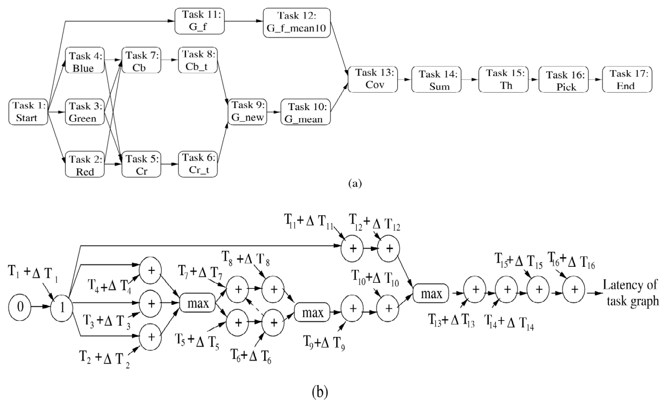

*Figure 2*: (a) Task graph and (b) Performance Model for the face detection algorithm

The procedure for finding the conditions that trigger architectural reconfiguration identifies execution time variations that are tolerable for a given architecture without exceeding the overall latency requirement. During execution, if execution time variations are larger than the found ranges, then architectural reconfiguration is initiated. The proposed algorithm analyzes different range values for the task execution times, and for each range it checks whether the overall system latency is still satisfied. Interval arithmetic is used to find the resulting ranges for system latency.

## 4. CASE STUDY

To illustrate the methodology, we refer to the face detection algorithm shown in Figure 2(a). This algorithm is part of a camera-based sensor architecture that we recently proposed for tracking applications [Weng,2004].

For the face detection algorithm, Figure 2(b) shows the PM for latency. The figure assumes that tasks 5, 6, 7 and 8 are executed in this order on a shared processor. The constant part of the PM includes all nodes and solid edges in Figure 2(b). *max* and *addition* nodes express constraints between start and end times of each task. For example, the outputs of max nodes define the start time of the corresponding task. The start time of a task has to be larger than the maximum of the end times of all its predecessors. Addition nodes express that the end time $T^{end}_i$ of task *i* is the sum of its start time $T^{start}_i$ and its execution time $T_i$. The variable part presents the relationship between latency and the design decisions taken during architectural optimization, like task partitioning to processors and task scheduling. In Figure 2(b), the variable part includes dashed arcs between the addition nodes for the end times of tasks, and the max nodes for their start times. Note that task execution times are expressed as $T_i + \Delta T_i$ ($T_i$ being the minimal execution time and $\Delta T_i$ is the execution time variation). Intervals $\Delta T_i$ will be used to find reconfiguration conditions.

*Table 1*: Sequence of resource sets

| | Adders | Multipliers | Active area ($mm^2$) | Implementations |
|---|---|---|---|---|
| Set 1 | 128 | 128 | 70 | Tasks 13-16 in HW; rest in SW |
| Set 2 | 512 | 128 | 76 | Tasks 12-16 in HW; rest in SW |
| Set 3 | 128 | 512 | 129 | Tasks 12-16 in HW; rest in SW |
| Set 4 | 512 | 512 | 136 | Tasks 12-16 in HW; rest in SW |
| Set 5 | 1024 | 1024 | 268 | Tasks 12-16 in HW; rest in SW |

*Table 2*: Reconfiguration conditions

| Taks | Set 1 | Set 2 | Set 3 | Set 4 | Set 5 |
|---|---|---|---|---|---|
| Cr | < 588 | < 425 | < 365 | < 365 | < 192 |
| $Cr_{th}$ | < 440 | 151-513 | 61-151 | 32-61 | < 32 |
| Cb | < 320 | 320-365 | 365-425 | 192-365 | < 192 |
| $Cb_{th}$ | < 440 | 440-513 | 61-151 | 32-61 | < 32 |
| Gmean | < 155 | 155-180 | 41-52 | 52-78 | < 52 |
| Gfmean | < 17 | < 20 | < 17 | < 21 | < 22 |
| Cov | < 301,500 | < 218,000 | < 255,590 | < 24,518 | < 65,536 |
| Sum | < 245,000 | < 218,000 | < 255,590 | 255,590-311,296 | 311,296 - 327,680 |
| Th | < 147 | < 170 | < 50 | < 61 | < 32 |
| Pick | < 282,600 | 282,600-326,860 | 282,,600-326,860 | 326,860-373,555 | 373,555 - 393,216 |

We used five resource sets in the case study. Table 1 presents the characteristics of the resources sets, the type and number functional units, and the corresponding ASIC area that is active for each set. Active area is used to qualitatively model the consumed energy, as lesser active hardware resources will decrease energy consumption. The last column presents the hardware-software partitioning results for the five different resource sets. Tasks 13, 14, 14 and 16 are placed in hardware considering their high execution times.

Set 1 represents the minimal resource set, and the task execution times are shown in Column 2 of Table 2. It shows that for certain task execution ranges, the architecture for Set 1 meets the fixed system latency constraint of 840k clock cycles. Note that this architecture allows a flexibility of about 11.5% meaning that the cumulative task execution time variations on the critical path should be percentage-wise less than the computed flexibility. Resource set 1 is not very flexible due to the fact that the usage of hardware resources is high. The second resource set corresponds to tighter timing constraints. In this case the task execution time belongs to the intervals shown in the third column of Table 2. The flexibility of the resources set is higher than for the previous case being around 1.33. Larger task execution time ranges can be accommodated in this case. Similarly, Table 2 shows the execution time ranges for resource sets 3, 4 and 5. Highest flexibility is achieved for Set 5, which also includes the most hardware resources.

During functioning, if any of the task execution time ranges is exceeded than an event is triggered, which is handled by the reconfiguration controller by activating 384 more adders when switching to the second resource set. Similarly, if any of the execution time ranges is exceeded then 384 multipliers are added to the architecture, while 384 of the adders are de-activated. The reconfigured architecture corresponds to resource set 3, which accommodates the execution time ranges shown in Column 4 of Table 2. Similar transitions exist from resource set 3 to resource set 4, and resource set 4 to resource set 5. Whenever execution times are relaxed below the specified time ranges, the controller switches the architecture back to the immediately "lower" resource set, such as from resource set 5 to resource set 4, and so on.

For example, resource set 1 is used as long as the execution time for task *Cr* is below the limit of 588 clocks, that of task *Cb* below the limit of 320 clocks, and so on. If the input data size increases (like due to larger images of interest that need to be processed) the execution time of Task *Pick* might go up beyond the limit of 282,600 clocks. In this situation, resource set 1 cannot meet the imposed latency constraint of 840k clock cycles. The reconfiguration controller detects this situations, and triggers a reconfiguration to resource set 2. Now, if the execution time of Task *Cov* is beyond 218,000 then depending on the value of Task *Pick*, the controller might select Set 1 or Set 3. If the time for *Pick* is below the limit of 282,600 clock cycles then set 1 is chosen, otherwise reconfiguration will correspond to set 3.

From the experimental result it is shown that Set 4 only takes around half of the hardware area as Set 5, whereas the flexibility remains almost the same (1.9 for set 4 compared to 2.0 for set 5). Therefore, it can be concluded that an upper limit exists for the flexibility: as the hardware area increases above the limit, the hardware area increment stops to gain performance enhancement. Comparing Set 2 and Set 3, it can been seen that the number of adders contributes more to the flexibility than the number of multipliers. This is due to the fact that the tasks in the face detection application have more additions than multiplications. Therefore, allocating more adders results in higher flexibility. The other advantage of adders is the much smaller area it has over the multipliers.

## 5. CONCLUSIONS

This paper presents a methodology for dynamic adaptation of sensor node architectures to improve the scalability of a sensor network. The method aids producing control algorithms for on-line adaptation, so that the node's architecture effectively addresses continuously changing operation conditions, like variable data rates and changing latency constraints. To illustrate the method, we discuss a case study on a camera based smart sensor node. We show that architecture adaptation is capable of employing the minimal set of resources needed for a given task, thus reducing the energy consumption of a node.

## REFERENCES

[Brooks,2003] R. R. Brooks, P. Ramanathan, A. M. Sayeed, "Distributed Target Classification and Tracking in Sensor Networks", January 2003.

[Borgatti,2003] M. Borgatti, F. Lertora, et al, "A Reconfigurable System Featuring Dynamically Extensible Embdded Microprocessor, FPGA, and Customizable I/O", *IEEE Journal Solid-State Circuits*, Vol. 38, No. 3, pp. 521-529, March 2003.

[Doboli,2001] A. Doboli, "Integrated Hardware-Software Co-Synthesis and High-Level Synthesis for Design of Embedded Systems under Power and Latency Constraints", *Proc. Design, Automation and Test in Europe Conference*, 2001.

[Estrin,2002] D. Estrin, D. Culler, K. Pister, G. Sukhatme, ``Connecting the Physical World with Pervasive Networks", *IEEE Pervasive Computing*, Vol. 1, No. 1, pp. 59-69, January-March 2002.

[Haubelt,2002] C. Haubelt, J. Teich, K. Richter, R. Ernst, "System Design for Flexibility", *Proc. of the Design, Automation and Test in Europe Conference*, 2002, pp. 854-861.

[Jones,2001] Jones, C.E., Sivalingam, K.M., Agrawal, P. and Chgen, J.C., "A Surevy of Eneregy Efficient Network Protocols for Wireless Networks", *Wireless Networks*, Vol,. 7, 2001, pp. 343-358.

[Kahn,2000] Kahn, J.M., Katz, R.H. and Pister, K.S.J., "Emerging Challenges: Mobile Networking for "Smart Dust", *Journal of Communication Networks*, vol. 2, no. 3, Sept. 2000.

[Stolfi,1997] J. Stolfi, H. de Figueiredo, "Self-Validated Numerical Methods and Applications", *Proc. of the 21st Brazilian Mathematics Colloquium*, July 1997.

[Weng,2004] Y. Weng, A. Doboli, "Smart Sensor Architecture Customized for Image Processing Applications", *Proc. of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, Toronto, 2004.