Scheduling Data Flow Programs in Hard Real-Time Environments

Renzo Davoli[1], Fabio Tamburini[2] and Luigi-Alberto Giachini[3]

[1] Dept. of Computer Science, University of Bologna, Italy.
[2] C.I.L.T.A., University of Bologna, Italy.
[3] Dept. of Mathematics, University of Bologna, Italy.

Abstract. Data Flow is a natural paradigm representing distributed programs able to express data and control dependencies between composing nodes. Our work concentrates on periodic Data-Flow programs having Hard Real-Time constraints. We propose a model and a string-based representation for a relevant class of Data-Flow programs and present a two-phase algorithm able to compute a schedule for a given set of programs to be executed. The first phase computes a feasible solution (mapping and time assignments) using a Simulated Annealing technique, the second phase takes the mapping as input and computes the optimal solution, in terms of resource allocation, using a recursive descent Quasi-Newton method.

1 Introduction

In this paper we focus on the class of Hard Real-Time (HRT) systems, those which may cause catastrophic effects in case of a timing deadline miss. In HRT program tailored systems, an off-line scheduling analysis [18] is necessary to guarantee the availability of processing resources when a critical situation arises. Dynamic on-line schedulers are not appropriate when dealing with HRT programs: given the finite amount of available processing resources, the system may not be able to honor the requests of the new entering program. In a HRT environment this case is equivalent to a timing failure, so it may lead to dangerous situations (e.g. plane crashes or nuclear disasters).

The processing environment we assume is multiprocessor and loosely coupled; a distributed system is a well-known example of this sort of architecture. The processors, for the purposes of this paper, are presumed to be homogeneous: a program progresses at the same speed on every system processor.

We present a model to express the schedulability problem for a set of Data-Flow (DF) periodic programs. We provide a two-phase algorithm to compute one of the solutions which may minimize the used processor quotas: in the first phase a simulated annealing process computes a feasible solution to the schedulability problem; in the second phase a gradient descent method, polynomial bound in time, is applied to further refine the solution. The second phase is useful to provide processing power to non real-time applications that may be present in the system and also to admit new HRT programs while the system is running.

The uncertainty introduced by the heuristic technique (Simulated Annealing) only affects the possibility of finding a solution or the quality of the solution found in terms of resources used, but in no case does it alter the real feasibility of the computed scheduling. In other words, if the off-line schedulability checker is able to compute a schedule, it may not be the best one, but it is feasible.

At the end of the paper we present an experimental results section. Some DF programs are given to show the effectiveness of our approach: a sort algorithm, a Fast Fourier transform and an image compression algorithm (Wavelet Transform [19]). Some instances and combinations of these programs are used to test the schedulability algorithm and to compare the solutions obtained with the optimal one.

2 Problem Statement

In the model we propose, programs are not atomic entities: they consist of nodes that may be executed concurrently, and hopefully in parallel, by different processors. The data communications and precedence relations between the nodes of a program are expressed using the deterministic Data Flow paradigm. In the DF paradigm, programs are represented by acyclic graphs, whose arcs embody both data communication and timing precedence relationships and whose nodes are procedures having functional behavior (inputs are known at activation time, results are given at the end, there is no communication in-between). This paradigm makes all the sequential and concurrent parts of the problem explicit, allowing the construction of RT schedules on the basis of the whole program timing specification (its period and deadline) and the computational cost (or weight) of its nodes. The node weight represents its maximum execution time, in the worst algorithmic case, on a single dedicated processor. This definition is particulary applicable to HRT problems with fixed deadlines [13].

The selected programming paradigm is a well-suited abstraction for parallel and distributed system applications. As an example, in [1], a similar approach is presented for distributed system programming together with a prototype programming interface and some examples of non-trivial programs.

The results presented in this paper refer to a specific subset of DF programs: the scatter-gather (SG) programs. These are created by the recursive application of two basic operators: a sequence operator and a concurrent execution operator. Formally a Scatter-Gather (SG) program has the following recursive definition:

- a single atomic computational node is an SG program;
- the sequential concatenation of two or more SG programs is an SG program;
- the concurrent execution of two or more SG programs is an SG program.

The principal advantage of this restriction is represented by the possibility of better minimizing or totally eliminating interferences between the different concurrent components. The related lack of generality, in our opinion, cannot be considered so important; the class of problems that can be described with such a model is not banal and represents a powerful platform to start from.

In our model the deadline of each activation coincides with the period end. Periodic programs may also have different periods and deadlines, but this case is less common in applications. In Sect. 7 we briefly explain how our method could be adapted to handle these cases.

Although our scheduling approach is tailored to periodic programs, it may deal with sporadic and event-driven programs as well, given that the maximum signalling frequency of the triggering events (for event-driven programs) and the maximum response time (for both) are known a priori. Moreover each program minimum activation delay must be greater than the time interval needed to perform its job (without exceeding its RT constraints). Given these conditions, in fact, it is possible to manage all programs as if they were periodic [12]; therefore, a knowledge of the requested parameters is necessary to limit the amount of resources to be allocated.

Checking the schedulability of a set of DF programs means not only providing a suitable mapping function but also evaluating how to distribute the response time of each program among its nodes. As for mapping, we regard this as a function having a logical set of nodes as its domain and a physical set of processors as its co-domain. From a theoretical point of view, the mapping problem consists of finding a partition of nodes leading to a feasible schedule. This is a search problem on all subsets of the node set. Clearly, an optimal solution can be found only by performing an exhaustive search on this set, involving an intractable $\mathcal{NP}$ problem: this operation is isomorphic to the N-knapsacks problem so it cannot be solved in polynomial time (unless $\mathcal{P} = \mathcal{NP}$). Moreover, changing the time assignment of a node affects the processor load for its execution: therefore, even if the mapping had been fixed a priori, a suitable time assignment would have to be computed. In [3] we have already presented a deterministic algorithm to compute the time assignment function, given a previously computed mapping between nodes and processors. Here we wish to extend the results by showing that Simulated Annealing techniques can be used to compute both the mapping function and the time/load assignment of the nodes at the same time. A deterministic algorithm can be used to refine the result.

3 Related Work

Our work is based upon the classical and well-known result of Liu and Layland [9]: Earliest Deadline First (EDF) scheduling is optimal for a set of indepedent periodic processes on a single processor. Schedulability checking in such a system is carried out by controlling that the processor is not overloaded. We extend the property along two independent directions: from a single processor to a multiprocessor loosely coupled environment and from unrelated processes to a set of executable entities with data and precedence constraints.

In a loosely coupled (private memory) environment, data exchange between tasks running on different processors is possible, while process migration should be avoided. This assumption is consistent with common implementations: a process migration involves either the real-time move of the task code or the avail-

ability of the code on all the processor memories and, in any case, the process state transfer. This operation would lead, in our opinion, to a waste of system resources and to a lower degree of time determinism. As a result there could be a clash with HRT requirements.

This characterization makes the classical results for shared memory systems (e.g. [11]) inapplicable to our model. In particular see the non-optimality result of Mok [12] for EDF scheduling in multiprocessor environments; it does not apply to our model because we run a set of independent EDF schedulers, one for each processor, with a pre-computed mapping, not a global EDF scheduler for all processors.

Several papers listed in the bibliography have similarities with our work or address different but related problems. Muntz and Coffman [13] presented a method to minimize the schedule length of a single rooted-tree structured program to be executed once (single shot, not periodic) in a multiprocessor environment.

The recent paper by Gerber, Hong and Saksena [6] addresses the problem of feasibly scheduling periodic processes with precedence constraints on a single processor. Their work is specifically tailored to satisfy end-to-end timing parameters of an application when data has to be managed by processes having different periods. Zhu, Lewis et al. [22] also address a similar problem: in their model a real-time program is described by a graph having two arc sets to represent data and precedence relationships separately. Each process has its own period and the operating environment is uniprocessor. Our work adds a new dimension to this topic of research: parallel execution. The model we present deals with a set of parallel periodic programs. In our opinion it is both essential and common for HRT programs to be able to perform heavy CPU-bursting computations. In such cases parallel execution is crucial. Moreover, the model could work as a basis on which to design a modular general purpose environment for (Hard) Real-Time applications.

Sih and Lee [16, 17], El-Rewini and Lewis [14] have already addressed the problem of scheduling graph structured programs onto a set of processors. They compute the minimal schedule length of a set of tasks given their precedence relationships using a non-preemptive policy. Their method could verify the consistency of a periodic execution by checking whether the minimal schedule length is shorter than the (single) requested period, but their consideration cannot be directly applied to an environment where several graph structured parallel programs having different periods are to be executed on a single parallel system. Our method addresses the latter problem: it searches for the feasible schedule that minimizes the processor usage while their search for minimal schedule generally leads to a bad resource allocation: all the processors are used more than necessary to respect the timing constraints. The scheduling policy is also different: we adopt a preemptive paradigm. We have to point out, however, that their methods take into account communication costs, while our current model does not specifically address that issue. We plan to extend our work in that direction.

Tindell, Burns and Wellings [20] address the problem of scheduling a set of tasks onto a parallel computer taking into account the data dependencies

between them. The scheduling paradigm is non-preemptive. They do not consider graph structured periodic programs but instead in their model each task has its own period. This fact leads to the same class of synchronization problems to compute the end-to-end timing of an application as discussed in the already cited papers by Gerber et al. [6] and Zhu, Lewis et al. [22].

Off-line scheduling is necessary to pre-allocate resources before starting the critical process control in order to give real guarantees. This approach is common in telecommunication models when Hard Real-Time requirements are needed. In this sense, our research has similarities with projects such as Tenet (Berkeley University) [4].

4 A Formal View

The basic problem we address is to obtain a sufficiency condition that allows the execution of a set of unrelated programs $JSET = \{J_1, ..., J_{NPGMS}\}$ having HRT requirements, on a set of processors $PSET = \{P_1, ..., P_{NPROC}\}$ in a safe way. The period of a program J_i is D_i. Each activation of J_i has to be completed within D_i time units; moreover, D_i is also the minimum time between two sequential activations of J_i.

All programs J_i consist of atomic execution entities named nodes. Different nodes, even if they belong to the same program, can be executed on different processors (and they generally are, when possibile, in order to achieve parallel execution). Let $NSET = \{n_1, ..., n_{NNODE}\}$ denote the set of all the node identifiers (of all programs). Each identifier is unique, i.e. it appears once in the whole set of programs. Let c_i be the computational cost (weight) of node n_i. c_i represents the time for node n_i to be executed on a single dedicated processor, regarding its algorithm worst case: the effective n_i computation time clearly depends on input data. It generally changes with different instances; it is non-negative (i.e. it may be zero), but in no case can it exceed c_i. c_i also includes communication costs. Processors are thought to be homogeneous, so the computational costs are independent of node mapping.

The precedence constraints among all the nodes of each single program J_i, due to data or control dependencies, are represented through a string S_i. The strings S_i belong to a language describing the graph's structure of SG programs. The SG DF language has the following grammar G:

$$G = (\Sigma_t, \Sigma_n, P, E)$$
$$\Sigma_t = \{'(', ')', '|'\} \cup NSET$$
$$\Sigma_n = \{E, F, T\}$$

$$P = \begin{cases} E \rightarrow E\ '|'\ F \\ E \rightarrow F \\ F \rightarrow FT \\ F \rightarrow T \\ T \rightarrow node \\ T \rightarrow '('\ E\ ')' \end{cases}$$

This language is mostly intuitive because it is based on the same symbology and semantics as Regular Expressions. It is naturally based on two primary operators: concurrent execution and sequence. All nodes are indicated with a unique name. Concurrent execution is a binary operator $'|'$, while sequence is

indicated by an implicit operator, i.e. sequential elements are simply juxtaposed. It uses round parentheses for grouping.

Figure 2 in Sect. 6 shows some examples of SG DF graphs and the corresponding strings for the problems analyzed in this paper.

To complete the necessary definition set, we have to introduce the notion of restriction. Having a string S that satisfies the grammar G, the restriction of S to a subset T, $|T| \leq NNODE$, is the result of substitution by ϵ of all node names not belonging to T, further reduced using the following rules:

$$\alpha\,'|'\,\epsilon \to \alpha, \quad \epsilon\,'|'\,\alpha \to \alpha, \quad \alpha\epsilon \to \alpha, \quad \epsilon\alpha \to \alpha, \quad \epsilon\,'|'\,\epsilon \to \epsilon, \quad \epsilon\epsilon \to \epsilon, \quad '('\,\epsilon\,')' \to \epsilon.$$

Intuitively, the restriction of a string S to the subset T represents a string which contains only members of T as nodes and whose precedence relationships are consistent with those of S[4]. This operator is used to make explicit, among all the nodes of a program that are mapped on the same processor, which ones are concurrent and which are in a sequential chain and therefore mutually exclusive.

Our schedulability problem can be described as a search (i) to build a mapping function between all nodes and the set of available processors and (ii) to assign an execution time to each node in the system, such that:

$\otimes$
1. Load specifications are positive.
2. All programs complete their jobs within their respective deadlines.
3. No processor is overloaded.
4. Each node is assigned exactly to one processor.

More formally, the goal is to find (if any) a vector $\overline{\lambda}$ of load-assignment values $\lambda_1, ..., \lambda_{NNODE}$ and a mapping vector $\overline{\mu} = (\mu_1, ..., \mu_{NNODE})$, $\overline{\mu_k} \in \{1, ..., NPROC\}$ that binds each node to a specific processor, such that:

$$\begin{cases} \lambda_k > 0 & \forall k = 1, ..., NNODE \\ Time(\overline{\lambda}, S_i) \leq D_i & \forall i = 1, ..., NPGMS \\ \sum_{i=1}^{NPGMS} Load(\overline{\lambda}, rmap(\overline{\mu}, S_i, P_j)) \leq 1 & \forall j = 1, ..., NPROC \end{cases} \tag{1}$$

Each element of the $\overline{\lambda}$ vector, λ_k, represents the amount of processor power reserved for the corresponding node k. Using the relation $x_k = c_k/\lambda_k$ to compute the deadline of each node it is possible to transform (1) to obtain a deadline vector $\overline{x}$ instead of a load-assignment vector $\overline{\lambda}$ (these two representations are perfectly equivalent).

The $rmap(\overline{\mu}, S, P_j)$ function is the restriction of S to the set $\{n_k \mid \mu_k = j\}$; it selects the program substructure consisting of the nodes that are mapped on the same processor P_j. The $Time$ function computes the maximum execution time for program J_i given a vector $\overline{\lambda}$ of node-load assignments: it adds the deadlines (x_k) of all sequential nodes and takes the maximum value between concurrent

[4] E.g. the restriction of $a(b|c)(d|ef)g$ to the set $T = \{a, b, d, e, g\}$ is $ab(d|e)g$.

ones. The *Load* function computes the maximum load for processor P_j executing the nodes of program J_i mapped on it. It adds the concurrent components and takes the maximum value of all sequential ones. Formal definitions of $Time$ and *Load* functions can be found in the appendix.

The approach we propose is to solve (1) in two different steps: as a single problem (even if it is $\mathcal{NP}$) by using a Simulated Annealing process, and then, given a pre-computed mapping, as a non-linear system of equations.

5 Optimization Procedure

5.1 Phase 1 - Feasible Solution Computation

Phase 1 has to produce a feasible solution to the problem (1). This is done by defining a function, often called energy function, that has its minimum when all the problem constraints are satisfied (this minimum is not necessarely unique). Searching for a minimum of the energy function means finding a feasible solution to the scheduling problem (1). The method we use was introduced by Ingber [7] [8] as a modification of the Simulated Annealing technique.

Energy Function Definition. Our problem (1) involves two kinds of unknowns:

- the $\overline{\lambda}$ vector containing the processor load assigned to each node,
- the $\overline{\mu}$ mapping vector between nodes and processors.

We represent these data as a single vector $\overline{\eta}$ composed of $2 \times NNODE$ variables. The values of each unknown, η_i, span in the range $]0,1]$. The first $NNODE$ elements, $\eta_1, ..., \eta_{NNODE}$, represent the processor ratios assigned to each node ($\overline{\lambda}$ vector), where 0 means no work and 1 means maximum load (every cycle of the corresponding processor is used by this node). The second $NNODE$ elements, $\eta_{NNODE+1}, ..., \eta_{2 \times NNODE}$, represent the mapping function ($\overline{\mu}$ vector). To obtain the exact processor used by node n_i we apply the function

$$\mu_i = \lfloor \eta_{i+NNODE} * NPROC \rfloor \ .$$

Given the above definitions the energy function can be written as:

$$E(\overline{\eta}) = E(\overline{\lambda} \cdot \overline{\mu}) = \alpha \sum_{i=1}^{NPGMS} \Phi(Time(\overline{\lambda}, i) - D_i) + \beta \sum_{j=1}^{NPROC} \Phi\left(\left(\sum_{i=1}^{NPGMS} Load(\overline{\lambda}, rmap(\overline{\mu}, S_i, P_j)) \right) - 1 \right) \quad (2)$$

where:

- $\Phi(z) = \begin{cases} 0 & for\ z \leq 0 \\ e^z - 1 & for\ z > 0 \end{cases}$
- α, β are constants used to balance the strength of constraints.

The function $E(\overline{\eta})$ defined in (2) has its minimum in $E(\overline{\eta}) = 0$ when all the problem constraints (1) are satisfied. (Note that the first constraint in $\otimes$ is implicitly met by our unknown definitions, while the fourth is fulfilled by the definition of the object $\overline{\mu}$ as a functional relationship).

Annealing Process. Adaptive Simulated Annealing (ASA) [7] [8] [15] is a global optimization algorithm. The result of the algorithm is the discovery of the best global fit of a non-linear, non-convex cost-function over a δ-dimensional space. ASA is an algorithm designed to search for optimal function value in a constrained state space. New states $\overline{\eta}$ are generated using the random variable y. y is derived from a uniform distribution $u \in U[0,1]$ using the function

$$y = sgn(u - 1/2)\ T_k[(1 + 1/T_k)^{|2u-1|} - 1]$$

where T_k is the current temperature. New η_i are then derived using the relation $\Delta\eta_i = y(B_i - A_i)$, where A_i and B_i are the limits of the i-th dimensional space; therefore newly generated values η_i are constrained by $\eta_i \in [A_i, B_i]$. Deriving new parameters according to this method leads to an exponential temperature scheduling, in particular:

$$T_k = T_0\ \exp(-\gamma\ k^{1/\delta})$$

where T_0 is the starting temperature and γ is a user-defined parameter to help system tuning. Newly generated states $\overline{\eta}$ are subjected to an acceptance probability function

$$h(\Delta E, T_k) = 1/(1 + \exp(\Delta E/T_k))\ .$$

In our scheduling problem the parameter space has dimension $\delta = 2 \times NNODE$, and each variable is bound into the interval $]0,1]$, so $A_i = 0$ and $B_i = 1$, $i = 1, ..., \delta$. At each step a variable η_i is randomly chosen, a new value η_i' is computed, according to the above rules, and accepted using the probability function h. Figure 1a shows the algorithm in detail.

This annealing schedule ensures that a global minimum of energy function can be obtained statistically (proven in [7]).

If the dimension of the sampled space becomes too high, the convergence process may be prohibitively slow and one cannot commit resources to properly sample the search space ergodically. Therefore Ingber [7] introduces a new parameter, called the quenching factor Q, to speed up annealing. Using quenching means modifying the annealing schedule as follows:

$$T_k = T_0\ \exp(-\gamma\ k^{Q/\delta})\ .$$

```
Annealing_Process()
begin
  Init η̄ with random values
  k := 1
  repeat
    T_k := T_0 exp(−γ k^{1/δ})
    η̄' := η̄
    i := Random(1, 2 × NNODE)
    u := Random(0, 1)
    y := Sgn(u − 1/2) T_k[(1 + 1/T_k)^{|2u−1|} − 1]
    η'_i := η'_i + y(B_i − A_i)
    Constrain η'_i into ]A_i, B_i]
    if (Accept(h(E(η̄') − E(η̄), T_k))) then η̄ := η̄'
    k := k + 1
  until (E(η̄) = 0) or (T_k < MinTemp)
  if (T_k < MinTemp) then return NotFound
                     else return η̄
end
```

Fig.1a. Simulated Annealing algorithm pseudo code. (The function $Random(a,b)$ extracts uniform random numbers into the interval $[a,b]$. Sgn is the sign function).

```
Optimize_Loads()
begin
  Initialize x̄
  repeat
    v̄ := ψ(x̄)
    J := { ∂ψ(x̄)_i / ∂x̄_j }_{i,j}
    Q := JJ^t
    Compute z̄ as the solution of Qz̄ + v̄ := 0
    Δx̄ := J^t z̄
    repeat
      toofar := false
      x̄' := x̄ + Δx̄
      if (x̄' is outside the domain) then
        toofar := true
        Δx̄ := Δx̄/2
    until (not toofar)
    x̄ := x̄'
  until (ψ(x̄) = 0)
  return x̄
end
```

Fig.1b. Newton's modified algorithm pseudo code (the function ψ is derived from functions F applying the time linear constraints to reduce the number of unknowns).

In this case we no longer have the guarantee of finding the optimal solution given in the ASA proof, but, for practical purposes, tuning the system using Q and γ parameters is quite easy and gives good results.

If the scheduling problem cannot be solved the annealing algorithm stops returning the constant $NotFound$, when the temperature reaches a pre-assigned minimum ($MinTemp$).

5.2 Phase 2 - Solution Refinement for Load Optimization

The goal of the second phase is to provide a solution for problem (1) which approximates the minimum of the processor utilization factor, given a pre-assigned mapping function.

Our approach is based on a transformation which regards the load consistency functions (third equation in (1)) as components of a non linear multi-dimensional function having the program time requirements (second equation in (1)) as linear constraints. The unknowns represent time values ($x_k = \frac{c_k}{\lambda_k}, k = 1, ..., NNODE$), so they are positive values (first equation in (1)). Any solution is a zero of function $F(\overline{\lambda}) = (F_1(\overline{\lambda}), ..., F_{NPROC}(\overline{\lambda}))$, representing the residual load

$$F_j(\overline{\lambda}) = 1 - \left(\sum_{i=1}^{NPGMS} Load(\overline{\lambda}, rmap(\overline{\mu}, S_i, P_j)) \right) - \lambda_{j+NNODE} \quad \forall j = 1, ..., NPROC$$

within the constrained space. The latter term $\lambda_{j+NNODE}$ is logically equivalent to an idle process added to the computation of each processor load. These "virtual"

single node programs, one for each processor, represent the unused quota of each processor. None of the (virtual or real) nodes can have a zero execution time, thus the method is only able to search for solutions of the schedulability problem that do not completely allocate any system processor. In other words, the solution space of this non-linear system of equations represents all the solutions for (1) with the added constraint

$$\sum_{i=1}^{NPGMS} Load(\overline{\lambda}, rmap(\overline{\mu}, S_i, P_j)) \neq 1 \quad \forall j = 1, ..., NPROC \ .$$

Given this approach, it is possible for an execution to terminate just before the deadline, e.g. if the input data fulfils the worst algorithmic case for every computational node. A dual approach which minimizes the result in terms of time (instead of load) should also be possible. It would lead to an isomorphic problem of the same complexity with no real practical interest: there is no need for a program to guarantee its termination earlier than requested in its worst case; it is better to have some unallocated processing power that can be assigned to new incoming processing requests.

Both *Time* and *Load* functions are continuous, but they do not have any more helpful properties for an analytical or numerical approach (e.g. they are not continuously differentiable), because they need the computation of maximum along some components (as shown in the appendix where functions *Time* and *Load* are defined).

It can be proved [2] that all (and only) the feasible assignments which do not completely allocate the processors can be computed as solutions of a non linear system taking the following form:

$$\begin{cases} x_1, \ldots, x_N > 0 \\ AX = L \\ M - BY - CX = 0 \end{cases}$$

where $X = (x_1, ..., x_N)$ and $Y = (\frac{c_1}{x_1}, ..., \frac{c_N}{x_N})$. A, B and C are constant matrix, L and M are constant vectors computed by the transformation algorithm. A has maximum rank. N is greater than $NNODE$ as "virtual" nodes has been added, but given $\overline{X} = (\overline{x}_1, ..., \overline{x}_N)$ a solution of the system, then the array $\overline{\lambda}$ ($\lambda_k = \frac{c_k}{x_k}, k = 1, ..., NNODE$) is a solution for (1). This transformation phase involves a complex set of moves. Reference may be made to [2] for a complete explanation of the method and for consistency proofs. We show here the basic concepts used in an intuitive way.

- Inequalities of time constraints (second equation in (1)) can be considered as equations. (Processor loads cannot be increased by increasing the time assigned to each node).
- Sequential substructures of a program mapped on the same processor have the same amount of processing power (execution time is not increased by augmenting the load assigned to each node). In this way, all the maximum functions involved in the *Load* function computation can be substituted by equations.

- Maximum functions that appear in the computation of the *Time* function can also be avoided by inserting another kind of "virtual" node. These nodes represent the difference of times assigned to concurrent paths.

This restructured problem can be addressed as the search for a zero of an under-determined non-linear system of equations subjects to linear constraints in its independent variables. Newton-like methods can be effectively applied to solve the problem because of their fast convergence rate. This class of resolution methods operates on domains which are open sets, from a topological point of view. This causes the extra constraint on the processor load (to exclude the boundary). It should also be possible to solve the problem on the processor complete allocation boundaries. Once stated that an internal solution cannot be found, boundaries might be reintroduced one by one, solving several isomorphic problems each with one more constraint and one less degree of freedom. This process would increase the computational complexity of the process without really increasing (in realistic situation) the probability of finding a solution.

The algorithm we use (Fig. 1b) is a slight modification of Newton's generalized method [5]. The solution of under-determined systems has received relatively little attention in the literature of recent decades. Local convergence results of Newton's generalized method (using the generalized inverse as an iteration matrix) can be found in [10]. In our case, starting from a point inside the constraints and whose *Load* values are all greater than 1, the function concavity and the negative divergence along all the domain boundary points lead the algorithm to converge to a function zero (if one exists).

6 Simulation Results

We verified our method in three different ways: by checking that the method had found an acceptable solution to (1), by investigating the computational complexity exhibited by our algorithms and by verifying the solution feasibility with respect to the optimal one.

To test the effectiveness of our approach in terms of solution effectiveness we carried out several simulations with different degrees of complexity. Some DF programs, such as QSort, FFT and Wavelet Transform, were used in various combinations and with different deadlines. Figure 2 outlines the problem parameters and the resulting $\overline{\lambda}$ vector and mapping function. The deadlines used are very close to the theoretical limits imposed by node costs. These limits were computed considering the program structures, the single node costs and the resources available in the system. In our test the limit deadlines were computed by hand, having a priori knowledge of the best mapping function. We chose to use these strict deadlines in order to experiment our method on a hard scheduling problem. Optimization procedures tends, when the constraints are very strict, to be trapped into local minima of the energy landscape. Experimenting the method on a limit situation is a good test to validate the effectiveness of such procedures. As we are confident that the method finds a feasible solution (if one exists) even for hard scheduling problems, we could state that a failure in

n_k	c_k	μ_k	λ_k	n_k	c_k	μ_k	λ_k	n_k	c_k	μ_k	λ_k
q0	1.0	3	0.9961	f0	1.0	1	0.9961	w0	1.0	5	0.9961
q1	1.0	3	0.9961	f1	4.0	1	0.5000	w1	5.0	4	0.5000
q2	1.0	0	0.9961	f2	4.0	2	0.5000	w2	5.0	4	0.5000
q3	3.0	3	0.5000	f3	4.0	2	0.5000	w3	5.0	5	0.5000
q4	3.0	3	0.5000	f4	4.0	1	0.5000	w4	5.0	5	0.5000
q5	3.0	0	0.5000	f5	4.0	1	0.5000	w5	5.0	5	0.9961
q6	3.0	0	0.5000	f6	4.0	2	0.5000	w6	5.0	4	0.9961
q7	1.0	3	0.9961	f7	4.0	2	0.5000	w7	5.0	5	0.9961
				f8	4.0	1	0.5000				
				f9	1.0	1	0.9961				
Total Time			9.0118	*Total Time*			18.0078	*Total Time*			21.0431
Qsort Deadline			9.0200	*FFT Deadline*			18.0200	*WT Deadline*			21.0450
Limit Dead.s			9.0000				18.0000				21.0000

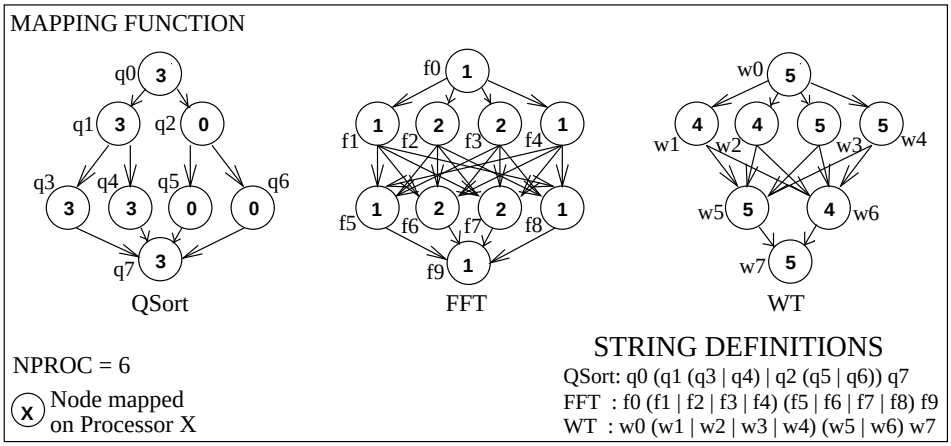

Fig. 2. Test involving different programs (an instance of QSort, an FFT and a Wavelet Transform.)

solution search will imply that there are not enough resources to accommodate the given set of programs. When the deadlines are not so strict the set of feasible solutions is wider, so an optimization method for resource usage has to be introduced, in order to allow new programs into the system.

The second test involved a heuristic evaluation of algorithm complexity. We investigated the increase in convergence time as a function of the number of programs involved in the system and the number of nodes forming the programs. These tests focused mainly on QSort-like programs. The results show that the convergence time grows as a quadratic function in relation to node number and program number. Figure 3 graphs this behavior: the test points were fitted to a second-degree polynomial function, obtaining very good interpolations. Figure 3a presents the schedule computation for a single QSort-like program on two processors. Figure 3b shows the correlation between the number of programs and

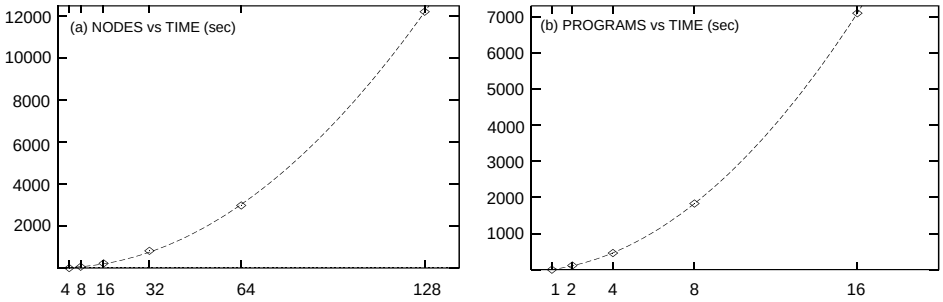

Fig. 3. Result graphs: number of nodes (a) and number of programs (b) vs. convergence time.

the time needed to compute a schedule. Each program was an eight-node QSort; the system had four processors. We should point out that it is statistically possible to find local minima of the optimization problem. Clearly, the probability of failure in the solution search increases with problem size, i.e. it depends on the number of nodes, programs and processors. In our test set, involving narrow solution spaces, ASA always reached the solution for all simulations. For significantly larger problems, either a local minimum can be found or the annealing factors have to be modified, affecting the complexity of the program.

The third test is a comparison between the solution found using our method and the optimal one, in terms of load balancing and optimization. In order to evaluate the effective improvement, in terms of load optimization achieved by the phase 2, Table 1 also presents the intermediate result, given by the phase 1. This is a feasible solution indeed, but it potentially wastes a lot of system resources, when compared with the optimal one and also with that provided by the complete method (phase 1 and phase 2). The solution obtained using the two-phase method is close to the optimal one. It is interesting to note that our algorithm exhibits the best behaviour when the problem is hard, involving strict deadlines. This is due to the fact that if the constraints force a narrow solution, then a good mapping has to be found by the annealing method. This mapping is equal to the optimal one (the only one allowed by constraint severity, without considering isomorphic mapping), so the load optimization process obtains very

QSort			*FFT*		
	Sol./Opt.			Sol./Opt.	
Deadline	1st Phase	All Phases	Deadline	1st Phase	All Phases
17.04	1.0020	1.0001	34.05	1.0015	1.0000
22.00	1.2647	1.0160	39.00	1.1425	1.0000
27.00	1.5734	1.0830	54.00	1.5170	1.0004
37.00	2.0329	1.0790	64.00	1.8420	1.0400

Table 1. Optimization tests.

good results, close to the optimal solution.

With regard to the parameter used to tune the method $(\alpha, \beta, \gamma, Q)$, we obtained the best results using the following combination of tuning parameters:

1. the *Time* and *Load* components in (2) (the constraints in (1)) are equally important in energy computation, so $\alpha = \beta$;
2. both parameters γ and Q, regulate the annealing process speed; to obtain good results we need to prevent the method from becoming trapped into local minima, so the temperature scheduling has to be slow enough. In particular, γ regulates the direct temperature scheduling, while Q adjusts the dimension of search space. Appropriate values for γ, to ensure a succesful cooling process, are contained in the interval $[10^{-4}, 10^{-3}]$. In our tests Q was similar to the dimension of the search space.

7 Conclusions

The schedulability problem for a set of SG programs can be expressed either as a non-linear set of equations or as an equivalent non-linear, non-convex function to be minimized. The independent variables, or the domain variables for the functional specification, are the load (or timing characteristics) and the mapping information of each node. Although this specification does not change the hard nature of the problem, it allows a number of challenging non-deterministic techniques to be applied. We showed that Adaptive Simulated Annealing can be used to give a feasible solution to the schedulability problem. Therefore the solution can be refined using a polynomial Quasi-Newton method to minimize resource requests. Our approach has produced encouraging results, both in terms of accuracy and complexity growth.

The model can be extended to deal with programs having different period and deadlines. A deadline smaller than the program period leads to a potential waste of resources, but the safety of our scheduling is guaranteed. The other situation, when the deadline is greater than program period, can be solved introducing more instances of the same program. In particular, if a program has a deadline D_i and a period W_i, with $D_i > W_i$, we can introduce $\lceil D_i/W_i \rceil$ instances of the same program J_i each with a period of $\lceil D_i/W_i \rceil W_i$. In this way we have a perfect use of system resources when D_i is a whole multiple of W_i and a potential waste of resources if not. Managing these situations as described above allows us to derive a sufficiency condition for the entire problem.

The proposed model does not explictely take into account the time consumed for context switch and data communication. However the model could be experimented, even with this approximation, by over-estimating the total time needed for data communication and context switches. As an example, given a TDMA network the communication time for the result produced by each node can be bound. Regarding the number of context switches, the following inequality holds:

$$num\_of\_cs \leq \sum_{j,i \in [1..NPGMS]; j \neq i} num\_of\_nodes(J_j) * \left\lceil \frac{W_i}{W_j} \right\rceil$$

where $num\_of\_nodes(J_j)$ is the number of nodes composing the program J_j. In this way network and context switch delays can be added to the computational cost c_i. This is a coarse approximation for many real applications, expecially when many programs are being executed in the system and many nodes are mapped on the same processor, exchanging large amounts of data. We plan to extend the methodology to compute the schedule of both processing and communication resources at the same time. The new method should take into account also the maximum time each program can take for context switches due to the interleaving with other programs.

We also plan to further improve non-deterministic methodology in order to address the schedulability problem for very large sets of programs and to extend the model to handle different processor speeds.

Appendix

Time and *Load* Functions

$Time(\overline{\lambda}, S)$ and $Load(\overline{\lambda}, S)$ functions are defined as the numerical results, respectively, $E.time$ and $E.load$ of the following Syntax Directed Definition (SDD) applied to S:

$E_1 \rightarrow E_2\ '\|'\ F$	$\{E_1.time = \max\{E_2.time, F.time\}; E_1.load = E_2.load + F.load\}$
$F_1 \rightarrow F_2 T$	$\{F_1.time = F_2.time + T.time; F_1.load = \max\{F_2.load, T.load\}\}$
$E \rightarrow F$	$\{E.time = F.time; E.load = F.load\}$
$F \rightarrow T$	$\{F.time = T.time; F.load = T.load\}$
$T \rightarrow n_k$	$\{T.time = c_k/\lambda_k; T.load = \lambda_k\}$
$T \rightarrow'\ ('\ E\ ')'$	$\{T.time = E.time; T.load = E.load\}$

References

1. Davoli, R., Giachini, L.A., Babaoglu, O., Amoroso, A., Alvisi, L.: Parallel Computing in Networks of Workstations with Paralex. IEEE Transactions on Parallel and Distributed Systems **7(4)** (1996) 371–384
2. Davoli, R., Giachini, L.A.: Schedulability checking of Data-Flow Tasks in Hard Real-Time Distributed Systems. UBLCS Technical Report Series **94-4** (1994)
3. Davoli, R., Giachini, L.A.: A Schedulability Algorithm for Data Flow Hard-Real-Time Distributed Programs. Proc. IFAC Real Time Programming, Lake Constance, (1994) 39–43
4. Ferrari, D., Verma, D.C.: A scheme for Real-Time Channel establishment in Wide Area Networks. IEEE Journal of Selected Areas in Communications **8(3)** (1990) 368–379
5. Fletcher, R.: Practical methods of optimization. J. Wiley and Sons (1986)
6. Gerber, R., Hong, S., Sakena, M.: Guaranteeing Real-Time Requirements With Resource Calibration of Periodic Processes. IEEE Trans. on Software Engineering **21(7)** (1995) 579–592
7. Ingber, L.: Very Fast Simulated Re-Annealing. Mathl. Comput. Modelling **12(8)** (1989) 967–973

8. Ingber, L.: Adaptive Simulated Annealing. Technical Report, Lester Ingber Research, McLean, VA (1993)
9. Liu, C.L., Layland, J.W.: Scheduling Algorithms for Multiprogramming in Hard-Real Time environment. Journal of ACM **20(1)** (1973) 46–61
10. Martinez, J.M.: Quasi-Newton methods for solving underdetermined nonlinear simultaneous equations. Journal of Computational and Applied Mathematics **34** (1991)
11. McNaughton, R.: Scheduling with Deadline and loss Functions. Management Science **6(1)** (1969) 1–12
12. Mok, A.K.: Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment. PhD Thesis, Dept. of Electrical Engeneering and Computer Science, MIT Cambridge-Mass. (1983)
13. Muntz, R.R., Coffman, E.G. Jr: Preemptive Scheduling of Real-Time Tasks on Multiprocessor Systems. Journal of the ACM **17(2)** (1970) 324–338
14. El-Rewini, H., Lewis, T.G.: Scheduling Parallel Program Tasks onto Arbitrary Target Machines. Journal of Parallel and Distributed Computing **9** (1990) 138–153
15. Rosen, B.: Function Optimization based on Advanced Simulated Annealing. IEEE Workshop on Physics and Computation - PhysComp 92 (1992) 289–293
16. Sih, G.C., Lee, E.A.: Declustering: A New Multiprocessing Scheduling Technique. IEEE Transaction on Parallel and Distributed Systems **4(6)** (1993) 625–637
17. Sih, G.C., Lee, E.A.: A Compile Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures. IEEE Transaction on Parallel and Distributed Systems **4(2)** (1993) 175–187
18. Stankovic, J.A., Spuri, M., Di Natale, M., Buttazzo, G.C.: Implications of Classical Scheduling Results for Real-Time Systems. IEEE Computer **28(6)** (1995) 16–25
19. Stollnitz, E.J., DeRose, T.D., Salesin, D.H.: Wavelets for Computer Graphics: A Primer. IEEE Computer Graphics and Application (1995) 76–84
20. Tindell, K., Burns, A., Wellings, A.: Allocating Hard Real Time Tasks (An NP-Hard Problem Made Easy). Journal of Real-Time Systems **4(2)** (1992) 145–165
21. Whao, W., Ramamritham, K., Stankovic, J.A.: Scheduling Tasks with Resource Requirements in Hard Real-Time Systems. IEEE Trans. on Software Engineering **13(5)** (1987) 564–577
22. Zhu, J., Lewis, T., Jackson, W., Wilson, R.: Scheduling In Hard Real-Time Applications. IEEE Software **12(3)** (1995) 54–63