Incorporating Imprecise Computation into System-Level Design of Application-Specific Heterogeneous Multiprocessors

Yosef G. Tirat-Gefen*
Dept. of EE-Systems (USC)
and Mentor Graphics Corp.
Wilsonville, OR 97070-7777

Diogenes C. Silva and Alice C. Parker
Dept. of EE-Systems
Univ. of Southern California (USC)
Los Angeles, CA 90089-2562

Abstract – This paper introduces a basic mixed integer-linear model (MILP) to design application-specific heterogeneous multiprocessors (ASHM) allowing imprecise computation of tasks executed in a non-preemptive mode. The proposed model was used in the development of a genetic algorithm integrated into the tool set MEGA that uses soft computing techniques for design of optimal/near-optimal ASHMs subject to constraints on performance, cost and output-data quality.

1 Introduction

The term *imprecise computation* was proposed by Liu et al. [7] to model real-time computations such as video processing, where it is possible to make a trade-off between quality of data (image) and computation time of particular video processing algorithms (e.g. video compression).

In the *non-preemptive imprecise computation* model assumed in this paper, each task S_i is divided in two parts: a *mandatory* (S_i^m) and *optional* (S_i^o), as seen in Figure 1. These parts correspond to subtasks subject to the following constraints:

i. The *optional* part must follow the *mandatory* part without interruption.

ii. They are allocated to the same processor.

iii. The *mandatory* part must be allowed to execute until completion.

iv. The *output* and *input* data volumes for the task are independent of the amount of processing allowed for the *optional* part.

*Formely known as J. C. DeSouza-Batista

This work was supported by the Advanced Research Projects Agency (ARPA) under contract no. 53-4503-9319 and monitored by the Federal Bureau of Investigation (FBI) under contract no. J-FBI-94-161 and the second author was partially funded by Brazilian National Council of Sciences (CNPq) under grant no. 201211/91.2.

DAC97, Anaheim, California
(c)1997 ACM 0-89791-920-3/97/06..$3.50

The result of a task is said to be a *precise* result if its optional part is allowed to compute until completion. By allowing *imprecise computation* of some tasks, it may be possible to meet hard time deadlines with a low implementation cost. The *imprecise execution* model assumes that the *quality* of the output data being produced by a task is a non-decreasing function of the amount of processing allowed for the optional part. The *imprecise computation* paradigm allows a extra fine-grain trade-off between overall data *quality*, performance and cost.

Typical applications of imprecise computation include *compression* algorithms such as *fractal compression*, digital signal processing transforms such as those used for finding the wavelet components of a non-periodic signal, or sub-band decomposition of video/audio signals. All these applications can model the *output data quality* as a function of the *processing time* expended by their respective algorithms.

Ergonomic studies have shown that human beings [1] are more sensitive to *audio quality* than to *video quality*. At the same time, *luminance quality* is more important than *chromatic quality* for video. These findings may be helpful when designing low-cost mass-production application-specific multiprocessor systems for real-time video/audio processing. Imprecise computation is able to represent these issues in a sound mathematical form.

Differently from Liu et al. [7], who developed scheduling algorithms for the preemptive mode of execution, this paper assumes that tasks are executed in a non-preemptive way in order to allow the development of low-cost application-specific AHSM implementations. The use of preemption would lead to some overhead of software and hardware, potentially increasing the overall cost of the design.

Consumer electronics products in areas such as video/audio processing are becoming increasily complex, and demanding high-performance implementations that utilize the available parallelism at a reasonable cost. Typical digital signal processing applications in video/audio can be specified as task-flow graphs, where different tasks usually have different associated algorithms that are best mapped to processors of different types. Application-specific heterogeneous multiprocessors (ASHMs) that mix custom and off-the-shelf processors are often the best option for such applications.

Mixed integer-linear programming (MILP) models allow a detailed representation of system behavior and provide a sound formal basis for the development of heuristics, as for example in the representation of the problem of concurrent task scheduling and processor allocation in ASHMs. However, attempts to use MILP to find optimal designs have been limited due to the computational cost (CPU-time) [10] which is prohibitive for large task-flow graphs. MILP solvers are also very sensitive to the linearization techniques used to derive a MILP model to be optimized.

Genetic algorithms are able to handle nonlinear constraints without need of linearization, which decrease the chances of non-convergence as well as allow handling of large task-flow graphs without a high computational cost. The price to be paid is that genetic algorithms are not always guaranteed to find an optimal solution.

This paper introduces an MILP model for system-level (task/processor level) design of ASHMs where tasks are non-preemptively executed and imprecise computation is allowed. This MILP model is used to derive a genetic algorithm for finding an optimal/near-optimal design.

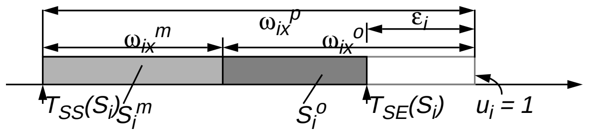

Figure 1: Modeling non-preemptive imprecise computation

2 Related work

The main results in *imprecise computation* theory are due to Liu et al. [3] [7] who developed polynomial time algorithms for optimal scheduling of preemptive tasks in homogeneous multiprocessors without communications costs. Ho et al. [5] proposed an approach to minimize the *total error*, where the *error* of a task being *imprecisely* executed is proportional to the amount of time that its *optional* part was not allowed to execute, i.e., the time still needed for its full completion. Polynomial time-optimal algorithms were derived for some instances of the problem of preemptive scheduling in homogeneous multiprocessors [7].

Chu et al. published one of the first mixed integer linear programming (MILP) models for the problem of simultaneous scheduling and allocation in existing multiprocessors. Recently the program SOS (Synthesis of Systems) [10], a compiler of MILP models, was developed. It takes a description of an task-flow graph, the processor library and some cost performance constraints, generating a output file with a MILP model to be optimized by available commercial MILP solvers. The SOS tool generates MILP models for the design of non-periodic (non-pipelined) heterogeneous multiprocessors, not supporting *imprecise computation.*

Genetic algorithms are becoming an important tool for solving the highly nonlinear problems related to system-level synthesis. Holand and his students at the University of Michigan in the late 1960s [8] introduced the first genetic algorithms by emulating the natural selection mechanism in biological systems, as discussed in Darwin's evolution theory. The use of genetic algorithms in optimization is well discussed by Michalewicz [8]. Research works involving the use of genetic algorithms to system-level synthesis problems are starting to be published, as for example Hou et al. [6] for scheduling of tasks in a homogeneous multiprocessor without communication costs; Wang et al. [9], Singh and Youssef [9], and Shroff [9] for scheduling of tasks in heterogeneous multiprocessors with communication costs but not allowing *cost versus performance* trade-off, i.e., all processors have the same cost, and Ravikumar and Gupta [11] for mapping of tasks into a reconfigurable homogeneous array processor without communication costs.

3 The MEGA tool set

This section describes the basic structure of the set of genetic algorithms implemented in the MEGA system, which is a soft-computing tool set for design of application-specific heterogeneous multiprocessors with non-negligible communication costs.

The version of MEGA discussed in this article supports a *non-periodic non-preemptive* mode of execution and two computational paradigms: *precise*, where all tasks are executed until completion, and *imprecise*, which allows the use of the *imprecise computation* model [7].

Task-flow graphs (TFGs) are used in MEGA to specify the application to be implemented as an ASHM. The vertices of the TFG correspond to tasks and the edges to data transfers. Each task S has a *performance trade-offs* list, which is a set of triples $(p_t, \omega^m_{p_t}(S), \omega^o_{p_t}(S))$, where p_t is a processor type and $\omega^m_{p_t}(S)$ $(\omega^o_{p_t}(S))$ is the computation time of the mandatory (optional) part of task S in a processor of type p_t,

$$\omega_{p_t}(S) = \omega^m_{p_t}(S) + \omega^o_{p_t}(S) \tag{1}$$

where $\omega_{p_t}(S) = \infty$ denotes that task S cannot be executed on processor of type p_t.

3.1 Chromosome representation for *precise computation*

MEGA has two types of genes:

i. *Discrete* genes whose domains are *isomorphic* to finite subsets of $\mathcal{N}$, the set of positive integer numbers. They are used to represent the allocation and *relative ordering* information.

ii. *Continuous* genes whose domains are closed *intervals* of $\mathcal{R}$, the set of real numbers. They correspond to the *timing* variables of the MILP model introduced by Prakash and Parker [10].

3.1.1 Allocation

The θ-genes and ϑ-genes correspond to the Boolean variables representing task allocation in the MILP model.

Definition 3.1 *The θ-genes set $\{\theta_1, \ldots, \theta_i, \ldots\}$ represents the* task *to* processor *instance mapping. $\theta_i = x$ denotes that task S_i is assigned to processor p_x.*

Definition 3.2 *The ϑ-genes set $\{\vartheta_1, \ldots, \vartheta_i, \ldots\}$ represents the* processor *to* processor type *mapping, where $\vartheta_x = t$ denotes that processor p_x is a processor of type t.*

In a similar way, π-genes and ϖ-genes correspond to the Boolean variables dealing with the allocation of non-local data transfers to buses.

3.1.2 Relative ordering

The α-genes (Gen_α) and Φ-genes (Gen_Φ) sets are directly related to the Boolean variable types $\alpha_{i\,j}$ and $\Phi_{i\,j}$ of the MILP model [10], where the $\alpha_{i\,j}$ ($\Phi_{i\,j}$) variables are only defined between parallel tasks (data transfers). The interpretation of these variables is the following:

i. α_{ij} is *true(false)* if S_i and S_j are in the same processor, and S_i finishes before (after) S_j.

ii. Φ_{ijrs} is *true* (*false*) if data-transfer $Dt_{ij} = S_i \rightarrow S_j$ is *parallel* to data-transfer $Dt_{rs} = S_r \rightarrow S_s$, Dt_{ij} and Dt_{rs} are in the same bus, and Dt_{ij} finishes before (after) Dt_{rs}.

3.1.3 Timing

Each timing variable of one of the MILP models discussed has an associated *continuous* domain gene. These genes, called *timing* genes, inherit all the properties of the timing variables. Therefore the same name used for a timing variable $T_x(\bullet)$ in the MILP model is used to identity its corresponding gene. For a given chromosome, their values are a function of the cost/performance parameters and the particular assignment to the discrete genes.

3.1.4 Genetic operators

MEGA uses *mutation* and *crossover* operators tailored to the system-level design problem, which were carefully designed in order to decrease the chances of creation of infeasible solutions.

3.1.5 Mutation

There are six types of genes in the chromosome representation used in MEGA for representing precise computation (Section 5 discusses the extension for the *imprecise paradigm*). Therefore, a mutation will correspond to a random change of one these genes.

Mutation of θ and ϑ genes

Let task S_i be assigned to processor p_j with type p_{t_k}, i.e. $\theta_i = p_j$ and $\vartheta_j = p_{t_k}$, for solution Sol_q.

Definition 3.3 *$\mathcal{FA}(S_i, Sol_q)$ is the set of available processors in solution Sol_q to which S_i can be reallocated without creating an* infeasibility*, where an infeasible choice can be one of the following:*

i. Processor p_y has a type to which S_i cannot be assigned.

ii. It is not possible to assign a bus (data link) between p_y and other processors to (from) which S_i sends (receives) data transfers due to *topologic constraints* of the interconnection network.

In both cases
$$p_y \notin \mathcal{FA}(S_i, Sol_q) \tag{2}$$

Definition 3.4 *The* mutation of the gene *θ_i is a random choice of a new allocation of S_i to one of the processors in $\mathcal{FA}(S_i, Sol_k) - p_j$, where p_j is the current processor to which S_i is assigned.*

Definition 3.5 *The set $\mathcal{FT}(p_j, Sol_q)$ is set of feasible types of processor p_j in solution Sol_q to which all tasks $S_u, \ldots, S_v$ assigned to p_j can execute, i.e.*

$$\mathcal{FT}(p_j, Sol_q) = \{\ types\ \ p_{t_i} \mid \vartheta_j = p_{t_i} \Longrightarrow \omega_{p_j}(S_u), \ldots, \omega_{p_j}(S_v) < \infty\} \tag{3}$$

Definition 3.6 *The* mutation of the gene *ϑ_j is a random choice of a new type for p_j from one of the available types in $\mathcal{FT}(p_j, Sol_q) - p_{t_k}$, where p_{t_k} is the current type.*

In order to keep this paper brief, the definition of the mutation of π and ϖ genes, very similar to the θ and ϑ cases, is omitted.

Mutation of genes in Gen_α and Gen_Φ sets

A valid mutation of a gene $\alpha_{i\,j} \in Gen_\alpha$ or $\Phi_{i\,j} \in Gen_\Phi$ is given by

$$\alpha_{i\,j} \longleftarrow 1 - \alpha_{i\,j} \qquad \Phi_{i\,j} \longleftarrow 1 - \Phi_{i\,j} \tag{4}$$

if this does not introduce a cycle in the corresponding overall schedule of tasks and data transfers of the task flow-graph G being mapped to an ASHM design.

Performing crossover

The operator *crossover* in MEGA treats a chromosome as a collection of *zones*, where each zone is associated with a subset of the Boolean variables from one of the MILP models for system-level design introduced by Prakash and Parker [10]. The minimal amount of genetic information exchanged during a crossover is a *zone.*

Task crossover

Definition 3.7 *Given two parent solutions (chromosomes) Sol_1 and Sol_2, a* task kernel *$\mathcal{K}_S(p_i, p_j) = \{S_a, \ldots S_z\}$ is a subset of the tasks in the TFG to be implemented, such that:*

i. All tasks in $\mathcal{K}_S(p_i, p_j)$ are respectively assigned to processors p_i and p_j in solutions Sol_1 and Sol_2, i.e.

$$\theta_a = \ldots = \theta_z = p_i \ \ in\ \ Sol_1 \quad and \quad \theta_a = \ldots = \theta_z = p_j \ \ in\ \ Sol_2 \tag{5}$$

ii. $\mathcal{K}_S(p_i, p_j)$ is maximal for the given pair of processors p_i and p_j, i.e. there is no other set $\mathcal{K}_S^{'}(p_i, p_j)$ whose elements also respect restriction 5 and $\mathcal{K}_S(p_i, p_j) \subset \mathcal{K}_S^{'}(p_i, p_j)$.

Definition 3.8 *Each task kernel $\mathcal{K}_S(p_i, p_j)$ defines a zone composed of genes $\theta_a = \ldots = \theta_z$ and the subset of α-genes $\mathcal{C}_\alpha(\mathcal{K}_S(p_i, p_j)) = \{\alpha_{i\,j} \mid S_i, S_j \in \mathcal{K}_S(p_i, p_j)$ and S_i is parallel ($//$) to $S_j\}$.*

Definition 3.9 *A* task crossover *is defined by the exchange of the corresponding zones of one or more task kernels chosen at random between two solutions (chromosome) Sol_1 and Sol_2.*

For the sake of brevity, the definition of the data-transfer *crossover operator*, which is similar to the task crossover operator, is omitted.

3.2 The graphical interface HERCULES

HERCULES is a GUI (Graphics User Interface) designed to act as a front-end to MEGA. It is composed of two major units: a dialog editor and a graphical editor. At least one input file must be defined: a library file, that specifies the available bus and processor parameters. A second file is optional and contains parameters and constraints for a particular MEGA run. The output is a set of two files: the task flow graph to be used by MEGA and and a graphical information text file for displaying the task flow.

4 An MILP model for imprecise computation

As seen in Figure 1, the *imprecise computation* paradigm assumes that each task S_i in the task flow graph can be divided in two: *a mandatory* part S_i^m and an *optional* one S_i^o, where the output data being generated by S_i has a *quality factor* Q_i, which is a non-decreasing function of the *utilization factor* u_i, $0 \leq u_i \leq 1$, where $u_i = 1$ means that S_i^o is executed until completion.

The *utilization factor* is the ratio of execution time S_i^o is allowed to execute over the time taken to completion. The key point is to trade the overall *quality factor* Q_{SYS} for a less expensive implementation.

$$Q_{SYS} = \sum_i Q_i(u_i) \tag{6}$$

where $Q_i(\bullet)$ is a non-decreasing function of the *utilization factor*, which might be nonlinear.

4.1 An MILP formulation

Assuming $Q_i(u_i) = k_i u_i$, this leads to

$$Q_{SYS} = \sum_i k_i u_i \tag{7}$$

The normalized overall quality factor is

$$NQ_{SYS} = \frac{\sum_i k_i u_i}{\sum_i k_i} \tag{8}$$

$$NQ_{SYS} = 1 \ \ for \ \ u_i = 1 \ \ \forall S_i \tag{9}$$

where k_i is a measure of the relative importance of the execution of S_i^o on the overall quality of output data to be produced by the multiprocessor system.
Let

$$\omega(S_i) = T_{SE}(S_i) - T_{SS}(S_i) \tag{10}$$

$T_{SE}(S_i)$ is the end of execution time for task S_i
$T_{SS}(S_i)$ is the start of execution time for task S_i

$$\omega^p_{S_i} = \sum_x \sigma_{i\,x} \omega^p_{i\,x} \tag{11}$$

Where $\omega^p_{i\,x}$ is the computation time of task S_i when allocated to processor x and allowed to execute till completion, i.e., the computation time of task S_i in the *precise computation* mode of execution.

$$\omega^p_{i\,x} = \omega^m_{i\,x} + \omega^o_{i\,x} \tag{12}$$

and $\omega^m_{i\,x}$ and $\omega^o_{i\,x}$ are the mandatory and optional computation times for task S_i when allocated to processor x, which are assumed to be constant parameters supplied by the designer.

$$u_i = \frac{\omega(S_i) - \sum_x \sigma_{i\,x} \omega^m_{i\,x}}{\sum_x \sigma_{i\,x} \omega^o_{i\,x}} \tag{13}$$

where $u_i = 1$ if $\sum_x \sigma_{i\,x} \omega^o_{i\,x} = 0$.
The utilization factor u_i will be a nonlinear function of the Boolean allocation variables $\sigma_{i\,x}$. In order to have a MILP model for the problem the quality factor of a task S_i can be reformulated as a function of the *processing error* ϵ_i.

$$\epsilon_i = \omega^p_{S_i} - \omega(S_i) \tag{14}$$

The quality factor will be some function $Q_i^e(\epsilon_i)$ of the error ϵ_i. The overall quality factor Q^e_{SYS} will be

$$Q^e_{SYS} = \sum_i Q_i^e(\epsilon_i) \tag{15}$$

Assuming

$$Q_i^e(\epsilon_i) = -k_i^\epsilon \ \epsilon_i \ \ ; k_i^\epsilon > 0 \tag{16}$$

The normalized value of Q^e_{SYS} will be

$$NQ^e_{SYS} = \frac{-\sum_i k_i^\epsilon \ \epsilon_i}{\sum_i k_i^\epsilon} = -\sum_i a_i \ \epsilon_i \tag{17}$$

$$where \quad a_i = \frac{k_i^\epsilon}{\sum_i k_i^\epsilon} \tag{18}$$

Therefore the MILP model for the case for imprecise computation of aperiodic non-preemptive tasks is given by

$$maximize \ \ NQ^e_{SYS} = -\sum_i \sum_x a_i \sigma_{i\,x} \omega^p_{i\,x}$$
$$+ \sum_i \sum_x a_i T_{SE}(S_i) - \sum_i \sum_x a_i T_{SS}(S_i) \tag{19}$$

subject to the other constraints of the MILP models for the nor-periodic non-preemptive mode of execution as discussed by Prakash and Parker [10].

5 A genetic algorithm approach

In order to allow the use of a genetic formulation for the problem of system-level design with *imprecise computation*, a set of a *utilization factor* genes is added to the chromosome representation used in MEGA for the *precise computation* paradigm.

Definition 5.1 *The* utilization factor u_i *gene takes real values in the range* $[0, 1]$ *and it is associated with the* utilization factor *of task* S_i,

A naive definition of the operator *mutation* for a real valued gene in the range $[a, b]$ would be a random choice of a real number in the interval. In a similar way, the crossover of two real valued genes can be defined as the exchange (swap) of their values.

Experimental results with an earlier version of MEGA using the naive definition of the *mutation* operator showed that the chances to reach an optimal/near-optimal design (solution), with a particular utilization factor u_i equal to 1.0 or 0.0, are minimal. Instead u_i would assume fractional values of the form $1.0 - \epsilon$ or ϵ, where ϵ is a small number, e.g. $\epsilon = 0.05$. In order to avoid that, the *mutation* operator is redefined using the concept of *trap function.*

Definition 5.2 *For a given a random number x in the interval $[a, b]$, $b > a$, and z, the* trap length*, where $2 * z < b - a$. The trap function $f_{trap}(x, z, a, b)$ is defined as*

$$f_{trap}(x, z, a, b) = \begin{cases} a & \text{if } a \leq x < a + z \\ \frac{(x-z-a)*(b-a)}{b-a-2*z} + a & \text{if } a + z \leq x < b - z \\ b & \text{if } b - z \leq x \leq b \end{cases} \quad (20)$$

5.1 Redefining mutation in MEGA

Definition 5.3 *The mutation of a* utilization factor gene *is defined as a change of its value to $f_{trap}(x, z, 0, 1)$, where x is a random real number in $[0, 1]$, and $0 \leq z < 0.5$*

5.2 Crossover

The same concepts of *task* and *data-transfer* crossovers, as defined in Section 3.1.5, apply to *imprecise computation.* However the associated zone of a task kernel is redefined as

Definition 5.4 *Each task kernel $\mathcal{K}_S(p_i, p_j) = \{S_a, \ldots S_z\}$ defines a zone composed by genes $\theta_a = \ldots = \theta_z$, $u_a, \ldots, u_z$ and $\{\alpha_{i\,j} \mid S_i, Sj \in \mathcal{K}_S(p_i, p_j) \;\; S_i//Sj\}$.*

5.2.1 Fitness value

The *fitness value* is a function of the cost, system latency (T_{SYS}) and overall data quality.

$$f_{fit}(x) = F_{max} - W_{cost} * Cost(x) - W_{T_{SYS}} * T_{SYS}(x) - W_{Q_{SYS}} * Q_{SYS}(x) \quad (21)$$

where F_{max} is a sufficient large number to ensure $f_{fit}(x) \geq 0$ for all possible solutions, $Cost(x)$ is the overall cost of the solution (design) x; and $Q_{SYS}(x)$ is a function of the values of utilization factor genes, not necessarily linear.

$$0 \leq W_{cost}, W_{T_{SYS}}, W_{Q_{SYS}} \quad (22)$$

5.3 Improving the quality by postprocessing

Given a value assignment for the *utilization factor genes* of a design implementing a directed acyclic task flow graph $G = (V, E)$, it is possible to recalculate the start and end times of tasks and data-transfers that are not in the critical-path in such a way that the system-latency is kept the same and the utilization factors of non-critical tasks are increased.

5.3.1 An overview of MEGA for *imprecise computation*

The following algorithm illustrates the extension of MEGA to the *imprecise computation* paradigm. The initial population is generated with all utilization factors equal to 1.0, due to the fact that previous experiments showed that it is difficult to reach an optimal solution in this region of the design space if MEGA allows utilization factors with values less than 1.0 in the initial population $\mathcal{P}(0)$.

Algorithm 5.1 *MEGA-ϵ - a genetic algorithm for system-level design of application specific multiprocessors allowing imprecise computation.*

```
Given a task flow graph $G = (V, E)$
Detect parallelism among tasks (data-transfers) using
transitive closure [4]
Generate initial population $\mathcal{P}(0)$ with all utilization factor
genes equal to 1.0
$t \longleftarrow 0$
repeat
    Perform mutation in a few solutions of $\mathcal{P}(t)$
    Perform crossover in a few solution pairs of $\mathcal{P}(t)$
    Find detailed timing of all solutions in $\mathcal{P}(t)$
     using a Bellman-Ford based ALAP algorithm.
    Improve utilization factor of non-critical
    tasks.
    Evaluate fitness of population $\mathcal{P}(t)$
    Scale and normalize fitness values
    Generate $\mathcal{P}(t+1)$ from $\mathcal{P}(t)$ by
    a predefined selection scheme [8]
    $t \longleftarrow t + 1$
until near-optimal or optimal design (solution) is found
```

6 Experimental Results

The tool MEGA was written in C, and it has approximately 7,500 line of code, incorporating both *precise* and *imprecise* computation paradigms. A set of benchmarks [10] (Figure 3) was adapted to evaluate the performance of MEGA. Tables 1, 2 and 3 provide detailed information about the tasks of the two task-flow graphs used as benchmarks as well as information regarding available processor and bus types. For the sake of simplicity, all data transfers are assumed to have a data volume equal to 1, with a negligible communication delay when *local*, where the overall delay of a non-local data-transfer is given by

$$Delay_{remote}(DT, bus) = \tau_{bus} + \frac{Volume(DT)}{s_{bus}} \quad (23)$$

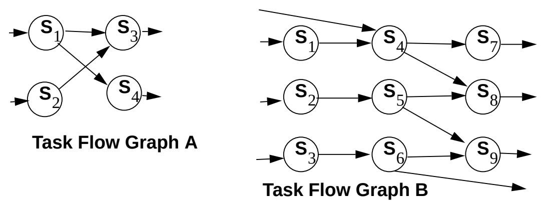

Figure 2: Task Flow Graphs

Processor		Tasks (S_i)	Computation time			
Type	.Cost		S_1	S_2	S_3	S_4
P_1	4	mandatory part	1.0	1.0	--	1.5
		optional part	0.0	0.0		1.5
P_2	5	mandatory part	1.5	1.0	1.3	0.5
		optional part	1.5	0.0	0.7	0.5
P_3	2	mandatory part	--	3.0	0.7	--
		optional part		0.0	0.3	
		Quality coefficients (k_i)	1.0	1.0	1.0	1.0

Table 1: Processor types - cost and performance information for TFG A

In all experiments, an *elitist* selection scheme [8] was used with mutation (p_m) and crossover (p_c) probabilities equal to

Processor		Tasks (S_i)	Computation time								
Type	.Cost		S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
P_1	4	mandatory part	1.0	1.0	0.5	1.0	1.0	1.0	2.0	-	0.7
		optional part	1.0	1.0	0.5	0.0	0.0	0.0	1.0		0.3
P_2	5	mandatory part	1.5	0.5	0.5	3.0	1.0	2.0	0.7	1.3	0.7
		optional part	1.5	0.5	0.5	0.0	0.0	0.0	0.3	0.7	0.3
P_3	2	mandatory part	0.5	0.5	1.0	--	3.0	1.0	2.7	0.7	2.0
		optional part	0.5	0.5	1.0		0.0	0.0	1.3	0.3	1.0
		Quality coefficients (k_i)	2.0	2.0	2.0	1.0	1.0	1.0	0.5	0.5	0.5

Table 2: Processor types - cost and performance information for TFG B

Bus			
Type	Cost	Speed	Latency
B_1	10	1.0	0.0
B_2	20	3.0	0.0

Table 3: Buses types - cost and performance information

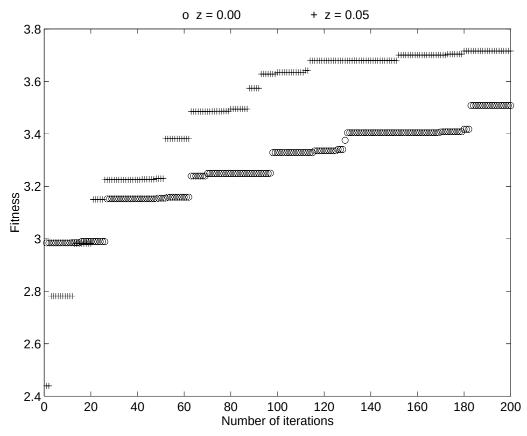

Figure 3: Effect of the trap length z (TFG B)

		Number of Processors			Number of Buses		Trade-off points		
Design	TFG	#P1	#P2	#P3	#B1	#B2	Latency	Cost	Quality
I	A	0	1	0	0	0	4.33	5.0	0.05
II	A	0	1	0	0	0	4.46	5.0	0.31
III	A	0	1	0	0	0	4.71	5.0	0.45
IV	A	0	1	0	0	0	7.0	5.0	1.0
V	A	1	1	1	1	1	2.67	41	0.83
VI	B	0	1	0	0	0	11.5	5.0	0.33
VII	B	1	0	1	1	0	7.31	16.0	0.45
VIII	B	1	0	1	1	1	5.33	36.0	0.95
IX	B	1	0	1	1	0	15.33	26.0	1.0
X	B	2	0	1	1	0	6.0	20.0	1.0

Table 4: Selected Designs

0.2 and a fixed size population of 100 chromosomes. Equation 10 is used in evaluating the overall *output-data quality*.

It can be seen from Figure 4 that the use of a small trap length z is helpful in speeding up the convergence to an optimal/near-optimal solution. This effect is more pronunciated for larger task-flow graphs (TFG B). Table 4 gives an idea of the design space for task-flow graphs A and B. Low quality solutions have associated low costs and latencies. An increase in quality requires more hardware resources. For a fixed cost, It is possible to have a lower latency by sacrificing the output-data quality. The run-time of MEGA was around a few minutes (< 5 min) in SUN-SPARC4 to generate all points in Figure 6 against hours using the SOS [10] approach, i.e optimization of MILP models restricted to the *precise computation paradigm*.

7 Conclusion

An MILP formulation was introduced modeling the system-level design problem when imprecise computation is incorporated. The extensions of MEGA based on this MILP approach were presented. A new gene set was introduced: the set of *utilization factor genes*, along with their respective mutation and crossover operators. The objective function was redefined by adding data quality to the possible trade-offs. A postprocessing heuristic to improve (increase) the utilization factors after performing detailed timing was proposed. The overall structure of MEGA for the imprecise computation paradigm was shown.

References

[1] BRAND, S., "The Media Lab : inventing the future at MIT", Kluwer Academic Pub., New York, 1988.

[2] CHU, W. W., HOLLAWAY, L.J. and EFE, K., "Task Allocation in Distributed Data Processing", *Computer*, Vol. 13, No. 11, pp. 57-69, Nov 1980.

[3] CHUNG, J.Y. , LIU, J. W. S. and LIN, K-J., "Scheduling Periodic Jobs that Allow Imprecise Results", *IEEE Tr. on Computers*, vol. 39, no. 9, pp. 1156-1174, Sept. 1990.

[4] CORMEN, T. H., LEISERSON, C. E. and RIVEST, R. L., "Introduction to Algorithms", McGraw-Hill Book Company, New York, 1989.

[5] HO, K., LEUNG, J. Y-T. and WEI, W-D., "Minimizing Maximum Weighted Error for Imprecise Computation Tasks", *Technical Report UNL-CSE-92-017*, Dept. of Computer Science and Eng., University of Nebraska, Lincoln, 1992.

[6] HOU, E.S.H, ANSARI, N. and REN, H., "A Genetic Algorithm for Multiprocessor Scheduling", *IEEE Tr. on Parallel and Dist. Systems*, Vol. 5, No. 2, pp. 113-120, Feb. 1994.

[7] LIU, J.W.S. LIN, K.-J., SHIH, W.-K., YU, A. C.-S., CHUNG, J.-Y. and ZHAO, W., "Algorithms for Scheduling Imprecise Computations", *IEEE Computer*, Vol. 24, No. 5, pp. 58-68, May 1991.

[8] MICHALEWICZ, Z., "Genetic Algorithms + Data Structures = Evolution Programs", Springer-Verlag, Berlin, 1994.

[9] PETERSON, L. (editor), Proc. of the 1996 Heterogeneous Computing Workshop, IEEE Computer Press, 1996.

[10] PRAKASH, S. and PARKER, A. C., "SOS: Synthesis of Application Specific Heterogeneous Multiprocessor Systems", *J. of Par. and Distributed Comp.*, No. 16, pp. 338-351, Dec. 1992.

[11] RAVIKUMAR, C. P. and GUPTA, A. K.,"Genetic Algorithm for mapping tasks onto a reconfigurable parallel processor", *IEE Proc. in Comp. Dig. Tech.*, vol. 142, no. 2, pp. 81-86, March 1995.