Multi-Criteria Optimization for Mapping Programs to Multi-Processors

Scott Cotton, Oded Maler
VERIMAG
Grenoble, France
Email: {Scott.Cotton, Oded.Maler}@imag.fr

Julien Legriel, Selma Saidi
VERIMAG, STMicroelectronics
Grenoble, France
Email: {Julien.Legriel, Selma.Saidi}@imag.fr

***Abstract*—Finding tradeoffs in design space is naturally formulated as a multicriteria optimization problem. In this paper, we model tradeoffs between communication cost and the balance of processor workloads for the problem of mapping applications to processors in a multicore environment. We formulate several *query strategies* for finding Pareto optimal and approximately Pareto optimal solutions to the mapping problem using a constraint solver as a time-bounded oracle. Each of the strategies directs the oracle through the search space in a different manner. We evaluate the efficiency of these strategies on a series of synthetic benchmarks, and on two industrial applications, a video noise reduction, and an image demosaic color filtering. The results indicate a significant tradeoff between precision and computation time, and a corresponding benefit to time-bounded queries.**

***Keywords*-Multi-criteria optimization; SMT solvers; Mapping ;**

I. Introduction

The ability to effectively map and schedule applications onto a multi-processor platform is a key factor in the success of future multi-core computers. We are interested in platforms where the multi-core *computation fabric* is intended to replace dedicated hardware in *data-intensive* applications such as video processing. For such applications, computation load distribution over processors *and* the workload it induces on the communication infra-structure are two main criteria in evaluating a mapping solution. These criteria are conflicting because as we move from a more centralized to a more parallel implementation we reduce the former and increase the latter. Computation workload and communication volume are just two out of numerous criteria considered in such design processes: memory size, energy consumption, price of components, are additional criteria that may be involved in evaluating and comparing design solutions.

Multi-criteria optimization problems [Ehr05], by definition, differ from single-criteria problems in the fact that there is no *optimal* solution(s) but rather a *set* of mutually-incomparable *efficient solutions* also known as *Pareto solutions*, which are solutions whose associated cost vectors are not *dominated* by any other feasible cost vector. Different Pareto solutions represent different design trade-offs and being able to compute a representative sample of these solutions is a valuable design aid [KTZ06]. Consequently there is a growing interest in adapting various algorithms and heuristics for solving hard optimization problems to the multi-criteria setting [Deb01], [MA04].

Several authors have addressed the mapping problem while optimizing multiple, potentially conflicting criteria. For instance DOL [TBHH07] and Sesame [EEP03] are frameworks for modeling and simulation of multiprocessors embedded systems based on *evolutionary algorithms* [Deb01]. They include a refinement loop based on successive optimization and simulation steps. The mapping step is implemented using SPEA [ZLT02] algorithm, one of the most popular genetic algorithms. The mapping problem has also been tackled using ILP [YHZ+09], [Ben96] or combination of ILP and constraint programming [RGB+06], sometimes using Benders Decomposition [BBGM05], a method that solves consecutive sub-problems to speedup the optimization process.

On the other hand, a great leap in performance has been achieved during the last decade for search-based methods for the satisfiability of Boolean formulae given in CNF form (SAT). Modern SAT solvers [ZM02] based on improvements of the DPLL procedures [DLL62] can now solve problems comprising of hundreds of thousands of variables and clauses and are used extensively to solve design and verification problems in hardware and software. SAT based approaches to solve design space exploration problem have been studied in [LYH+08]. Also in [SLHT06] the authors propose to introduce satisfiability techniques into a genetic algorithm.

Building upon the progress in SAT solvers, new tools emerged that extend them to handle mixed constraints involving both logic and numerical predicates . Some of these SMT (SAT modulo theories) solvers have achieved very good performance, for example for solving Boolean combinations of linear constraints [DdM06], [dMB08], which suggests that they can serve as the computational back-end for hard optimization problems such as mapping and scheduling. In [LGCM10] we developed a methodology for approximating the Pareto front by performing a multi-dimensional extension of binary search over the cost space, submitting queries to an external solver which serves as an oracle for the set of feasible costs. One advantage of using a solver in this context is the ability to obtain bounds on the

approximation error, expressed as the distance between the solution found and the actual Pareto front. Future integration of such techniques in the design flow (or even in the compilation flow) of embedded systems depends crucially on their performance.

In this work we perform a comparative study of the efficiency of different algorithms for finding Pareto optimal, and approximately Pareto optimal solutions for multiprocessors mapping problems evaluated according to the two above mentioned criteria: computation load balancing, and minimization of communications. The algorithms make use of an oracle for the set of feasible costs. Since testing membership in feasible cost space can be computationally hard, we introduce simple variants which make use of a time-bounded oracle.

The main difference between these algorithms is in the way they guide the solver toward unexplored parts of the cost space:

1) *Under-approximation with refinement*. These methods start with an under-approximation of the set of Pareto points and iteratively refine it by directing the solver to look for solutions in parts of the cost space which are incomparable with the solutions found. A new solution point from an incomparable part of the cost space is then further improved until a Pareto point is found, subject to time constraints. We describe two such methods which differ in how much control the solver has in directing the search toward incomparable points in the cost space.
2) *Distance reduction*. Variants of the algorithm of [LGCM10] which can produce an ϵ-approximation of the Pareto front. The algorithm maintains the *distance* between over and under-approximations of the Pareto front and guides the search as to minimize this distance. We consider variants of this algorithm which bias the search towards finding satisfiable points, since queries which minimize the distance tend to be hard and having satisfiable points is preferable to identifying unsatisfiable portions of the cost space in the intended application.

We have implemented these strategies, using the SMT solver Z3 [dMB08] as a time bounded oracle, and tested them against two classes of examples, synthetic task-data graphs generated by the TGFF tool [DRW98] and two realistic task graph derived from an implementation of the TMNR procedure for video temporal mosquito noise filtering [FLR00] and image demosaic color filtering application. As an execution platform we have chosen a data flow distributed memory architecture xSTream [BCA+09], provided by STMicroelectronics. Processors in this architecture are connected through an on-chip communication network, with a *Spidergon* topology [CLM+04] with a diameter $d = N/4$, N being the number of processors.

From a computational perspective our results show that there is a significant tradeoff between precision and computation time: individual queries near the optimal curve are hard and so it pays off to use time-bounded strategies which distribute more evenly the amount of work over the Pareto front.

The rest of the paper is organized as follows. In Section II we define task-data graphs, execution platforms and the mapping problem and then recall some terminology related to multicriteria optimization. Section III introduces multicriteria optimization using oracles and describes the three search space strategies that we implemented. Section IV is devoted to the description of the experimental setting, the implementation of the algorithms and the results, while Section V concludes.

II. Preliminaries

A. Task-Data Graphs and their Mapping

The mapping problem we model is a simplified form of the problem of parallelizing code, suitable for coarse-grained execution models. Our starting point is an application written in a dataflow style, consisting of components such as blocks and filters with well-defined precedence constraints where, in principle, any two tasks that do not precede each other can be executed in parallel. Hence it is natural to express an application as a task-data graph, in a manner similar to other formalisms such as synchronous data flow graphs [LM87] or streaming languages [TKA02], [BFH+04].

Definition II.1 (Task-Data Graphs)**.** *A task-data graph is a tuple $G = (P, d, v)$ where P is a finite set of tasks, $d : P \to \mathbb{N}$ indicates the amount of work associated with each task and $v : P \times P \to \mathbb{N} \cup \{\bot\}$ indicates the amount of data communicated between each pair of tasks.*

The amount of work $d(p)$ in a task p can be expressed in number of instructions or derived from application profiling. The work divided by processor speed gives the duration of the task. For convenience, we assume here a fixed speed of 1 for each processor so that $d(p)$ can be equated with task duration.[1] During each invocation, task p sends $v(p, p')$ data to any other task p' that it precedes. For the moment we do not make a distinction between sending all the data upon *termination* and sending data progressively *during* execution. We make here a distinction between $v(p, p') = 0$, where p is a direct predecessor of p' but the amount of data is negligible, and $v(p, p') = \bot$ where p is not a direct predecessor of p'.

We assume that the graph is acyclic, i.e. that there is no precedence or data communication path from any task to itself. The streaming aspect of the application is best modeled not as loops in the task-graph but rather as an input generator which provides instances of the task graph

[1] In [LM10] we handle scheduling problems on a configurable architecture with processors of varying speeds.

following some timing constraint. In [LM10] we have shown that such a periodic problem can be reduced to an acyclic task graph by unfolding a finite number of instances.

Definition II.2 (Execution Platform)**.** *An execution platform is a tuple $E = (M, N, b, \rho)$ where M is a finite number of processors, $N \subseteq M \times M$ is a set of (physical) communication channels all with the same bandwidth b (bits/second), and ρ is a routing function $\rho : M \times M \to N^*$ which assigns to every pair (m, m') of processors an acyclic path $(m, m_1), (m_1, m_2), \ldots (m_k, m')$ in the network. Clearly $\rho(m, m) = \epsilon$.*

By abuse of notation we also refer to the path as a *set* of channels and say that $(m_1, m_2) \in \rho(m, m')$ if (m_1, m_2) appears in the path.

Definition II.3 (Mapping)**.** *Given a graph G and an architecture E, a mapping is a function $\mu : P \to M$ with $\mu(p) = m$ meaning the task p executes on processor m.*

A mapping induces a workload on each processor, that is, the sum of execution times of the tasks that run on it. Likewise it induces a workload on the communication channels, the amount of data sent at each invocation of G over channel (m, m') is the sum of $v(p, p')$ over all pairs of tasks such that (m, m') is part of the path between $\mu(p)$ and $\mu(p')$. Let us define these workloads. For processors we have:

$$W(m) = \sum_{p:\mu(p)=m} d(p).$$

If the workload is perfectly balanced between processors, all processors have the same workload:

$$W^\star = \sum_{p \in P} d(p)/|M|.$$

For channels we first define the pairs of tasks that use a channel:

$$U(m, m') = \{(p, p') : (m, m') \in \rho(\mu(p), \mu(p'))\}$$

and then let

$$W(m, m') = \sum_{(p,p') \in U(m,m')} v(p, p').$$

Given a mapping μ, we define the two cost functions to optimize: workload distribution balance and communication cost.

Definition II.4 (Workload Distribution Balance)**.** *This function measures the difference in the application workload distribution over the processors, compared to a perfectly balanced distribution:*

$$\Delta = \sum_{m \in M} |W(m) - W^\star|$$

Definition II.5 (Communication Cost)**.** *The communication cost induced by the task distribution over processors is evaluated as the sum of volume of communicating tasks mapped on different processors, multiplied by the length of the path between these two processors:*

$$C = \sum_{(p,p'):\mu(p) \neq \mu(p)} v(p, p') \times |\rho(\mu(p), \mu(p'))|$$

Remark The model we present is not intended for precise scheduling; rather we expect workload distribution to give a reasonable estimation of execution time for the targeted systems. In particular, the systems are data-driven and we assume different instantiations of a task graph can be pipelined, resulting in an overall schedule with little idle time for any processor. In such an environment, optimal workload distribution should closely approximate optimal throughput. Also, the model we present here does not take into account the *identity* of data items. As a result, there is some potential imprecision in the communication cost for multi-hop architectures: suppose p sends the same data to p' and p'', and suppose that they are mapped to processors m, m' and m'', respectively. If the path from m to m'' goes through m' then we may count the load of (m, m') twice.

B. Multi-Criteria Optimization

Constrained optimization problems are often specified in the form

$$\min c(x) \text{ s.t. } \phi(x)$$

where x is a vector of decision variables, ϕ is a set of constraints on the variables that define which solution is considered feasible and c is a cost function defined over the decision variables. We prefer to reformulate the problem by moving costs to the constraint side, that is, letting $\phi(x, c)$ denote the fact that x is a feasible solution whose cost is c. Hence the optimum is

$$\min\{c : \exists x \ \phi(x, c)\}$$

In the case of multi-criteria optimization (MCO), c is a d-dimensional vector $(c_1, \ldots c_d)$ which ranges over a bounded cost space C. The set C is a lattice with a partial-order relation $\preceq$ defined as follows:

$$s \preceq s' \equiv \forall i \ s_i \leq s'_i \tag{1}$$

Pairs of points such that $s \not\preceq s'$ and $s' \not\preceq s$ are said to be *incomparable*, denoted by $s||s'$. The strict version of $\preceq$ is

$$s \prec s' \equiv s \leq s' \wedge \exists j \ s_j < s'_j \tag{2}$$

meaning that s strictly improves upon s' in at least one dimension without being worse on the others. In this case we say that s *dominates* s'.

A point s in a subset $S \subseteq C$ is *minimal* with respect to S if it is not dominated by any other point in S, and is *maximal* if it does not dominate any point in S. We denote the sets of minimal and maximal elements of S by $\underline{S}$ and $\overline{S}$, respectively. We say that a set S of points is domination-free

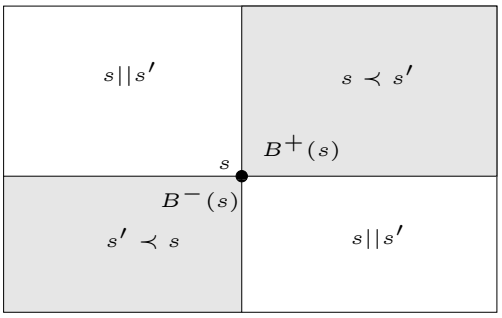


Figure 1. A point s in a two-dimensional cost space and its backward and forward cones.

if all pairs of elements $s, s' \in S$ are incomparable, which is true by definition for $\underline{S}$ and $\overline{S}$. The domination relation associates with a point s two *rectangular cones* $B^+(s)$ and $B^-(s)$ consisting of points dominated by (resp. dominating) s:

$$B^-(s) = \{s' \in C, s' \prec s\} \quad \text{and} \quad B^+(s) = \{s' \in C, s \prec s'\}.$$

These notions are illustrated in Figure 1.

III. Constrained MCO with Oracles

MCO problems are often solved in generic metaheuristic frameworks such as genetic algorithms [Deb01], [ZT99]. By contrast, we take the approach of issuing existential queries to a time-bounded oracle. This approach is convenient for problems which are naturally formulated in terms of constraints, as is the case with the execution model described in Section II-A.

In the single criterium case, it is straightforward to define a sequence of existential queries of the form $\exists x.\phi(x) \wedge c(x) \leq a$ to arrive at an optimal solution: a binary search over the range of c will converge to the minimum cost, with unsatisfiable queries bounding c from below while satisfiable queries bound c from above. But in the multi-criteria case, the problem is more complicated. First, every satisfiable query bounds a forward cone in the cost space while every unsatisfiable query bounds a backward cone. Second, the query sequence must give balanced coverage to the underlying curve of tradeoffs, even if some objective functions are harder to minimize than others. As a result, there are more choices for guiding the solver, both in terms of search directions and in terms of query time allocation.

Below, we describe the *space search* strategies. Each of these strategies may be parameterized by a per-query time limit on the underlying oracle. Since our goal is to find good sets of satisfiable points as quickly as possible, rather than to arrive at a *measurable* approximation as in [LGCM10], each search strategy treats a query which times out in exactly the same way as the case where the oracle indicated that the query was unsatisfiable. Thus the oracle is guaranteed to underapproximate the feasible cost space. Generally, the underapproximation becomes more precise as the time bound increases.

In the following we describe search strategies which explore regions of feasible cost space, or which explore regions of approximate feasible cost in the case where the oracle is time-bounded. The cost space is bounded from above by (w^*, c^*h^*) where w^* (resp. c^*) is the total work (resp. communication) associated with a task graph and $h^* = \max_{m,m'} |\rho(m, m')|$ is the longest path between any two processors. We also assume the oracle gives a satisfying assignment if it answers yes to a query.

A. Under-Approximation with Refinement

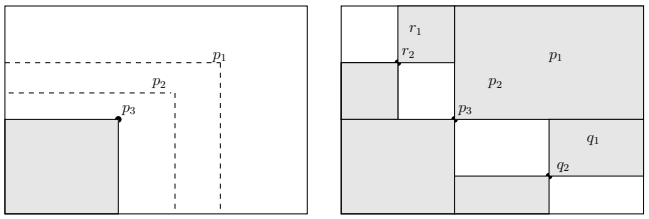
The first class of search strategies we consider is based on iteratively refining an underapproximation of the forward cone of the Pareto optimal points. Let P denote a mutually non-dominating set of feasible points. The strategies look for a refinement of $B^+(P)$ in the form of a set P' of mutually non-dominating points with $B^+(P') \supset B^+(P)$. Such a refinement is accomplished with a sequence of queries to the oracle of the form

$$\exists x.\phi(x) \wedge c(x) \prec r$$

For the moment, let us assume the sequence begins with a feasible point r. The query $\exists x.\phi(x) \wedge c(x) \prec r$ either gives a satisfying point $r' \prec r$ or indicates the query is unsatisfiable. If the query is unsatisfiable, r is Pareto optimal and the sequence stops. Otherwise, the sequence continues with the query $\exists x.\phi(x) \wedge c(x) \prec r'$. A maximal such sequence $(r_1, r_2, \ldots, r_k)$ gives a Pareto optimal point r_k.

The variants we consider differ only in how an initial point r_1 is chosen for each query sequence. For all the variants, the initial point of the initial sequence is determined by the oracle and simply falls at some unknown point in the cost space. Subsequent initial points of query sequences are chosen from the set of points incomparable to all points in P. As illustrated in Figure 2, this set of points is a union of rectangles, and we denote it $I(P)$.

1) A *union strategy* in which the queries ask the oracle for a feasible point in $I(P)$ as a disjunction over all the constraints defining rectangles in $I(P)$. This is a very indirect guidance, it just adds constraints that reduce the set of feasible solutions of the problem, but the effect is very solver-dependent.
2) A *maximum rectangle strategy* in which the queries ask the oracle for a feasible point in a largest rectangle in $I(P)$. If a largest rectangle is infeasible, subsequent queries will ask for a next-largest until either a feasible point is found or the set P is the Pareto front of the feasible cost space. Here, it is possible to guide the oracle so that it queries parts of the search space about which little is known.



(a) (b)

Figure 2. A series of runs from an under-approximation refinement strategy. (a) The first query sequence (p_1, p_2, p_3) gives the first pareto points p_3; (b) Subsequent query sequences (q_1, q_2) and (r_1, r_2) give other Pareto points q_2 and r_2. The set of points incomparable to any Pareto point in $P = \{p_3, q_2, r_2\}$, denoted $I(P)$, is the non-shaded region and takes the form of a union of rectangles. In both figures, the whole box represents the cost space C.

B. Distance Reduction

The basic idea of these search strategies, based on [LGCM10], is to maintain both over- and under-approximations of the dominated set of feasible points and to guide the search by reducing the *difference* between the two sets. As above, an under-approximation P is maintained in the form of a set of mutually non-dominating feasible points in the cost space. An over-approximation $\bar{P} = C \setminus B^-(U)$ subtracts an infeasible region from the cost space. Like $B^+(P)$, $B^-(U)$ is represented in the form of a set of mutually non-dominating points. However, every $u \in U$ has the property that $\forall u' \preceq u$, $\phi(x) \wedge c(x) = u'$ is unsatisfiable.

These strategies pose queries of the form $\exists x.\phi(x) \wedge c(x) \preceq s$ where

$$s \in C \setminus (B^+(P) \cup B^-(U))$$

After each query, either U or P is updated: A satisfiable result giving a witness w replaces P with the mutually non-dominating subset of $P \cup \{w\}$ and likewise an unsatisfiable result replaces U with the mutually non-dominating subset of $U \cup \{s\}$. All the strategies are based on identifying $u^* \in B^-(U)$, the point of U which is the furthest away from P. The distance between u^* to P is defined as the distance of u^* from its closest point $p^* \in P$, and the distance between $p \in P$ and $u \in U$ is defined as

$$\max_i (u_i \dot{-} p_i).$$

The strategies we consider here differ in how they use u^* whose distance from P is d to choose s. Let $\mathbf{d}$ denote the vector $(d, \ldots, d)$. We consider three strategies:

1) Binary search : query with $s = u^* + \frac{1}{2}\mathbf{d}$
2) Bias towards feasible : query with $s = u^* + \frac{3}{4}\mathbf{d}$
3) Randomly : ask $u^* + r\mathbf{d}$ where r is randomly chosen in $(0, 1)$

These notions are illustrated in Figure 3.

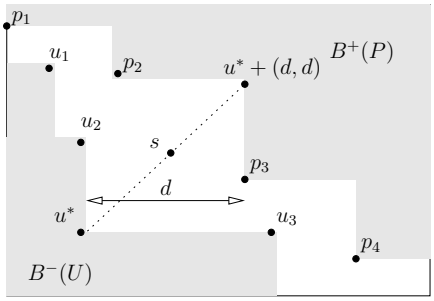


Figure 3. A figure representing over-under approximation based strategies. The whole rectangle represents the feasible cost space C. The shaded region labelled $B^+(P)$ is the forward closure of $P = \{p_1, p_2, p_3, p_4\}$ while the shaded region labelled $B^-(U)$ is the backward closure of $U = \{u_1, u_2, u_3\}$. The underapproximation of the forward closure of the feasible cost space is $B^+(P)$ while the overapproximation is $C \setminus B^-(U)$. Query points are chosen on the line segment between u^* and $u^* + (d, d)$, in order to reduce the distance of u^* (from which a Pareto point can be arbitrarily close) to P. Here, distance is measured as the maximal deterioration between u^* and p^*: if $u^* = (w_u, c_u)$ and $p^* = (w_p, c_p)$, then the distance is $d = \max(w_p - w_u, c_p - c_u)$.

IV. Implementation and Experiments

Our experimental problems consisted of a range of randomly generated task graphs and two industrial problems. The task graphs were generated with TGFF [DRW98]. The industrial problems are image and video processing applications: one noise reduction filter named TMNR [FLR00] and one demosaic color filter. Note that the major goal of this study is to explore the computational capabilities of the various search strategies, not to solve a specific concrete problem – this would require a more refined modeling of the application and the architecture.

TMNR is part of the image quality improvement process performed after video decoding, it reduces video temporal mosquito noise which appears as fluctuations of luminance and/or chrominance around sharp edges of moving objects. Given the current and previous frames, the algorithm assigns a motion value to each pixel based on neighboring pixels, and then combines the current frame, and the previous filtered frame to reduce the temporal mosquito noise. We build a task-data graph of 14 tasks, see figure4 based on the functional decomposition of a hardware implementation of the algorithm. The computation and communication weights are computed according to the basic computation unit of the algorithm, which is a sliding window of 10×10 pixels. We assume that by mapping this task-data graph for the basic block unit, the rest of the image will be processed in a pipelined manner.

Demosaicing is an image processing application for color filtering in the RGB color model, in which each pixel has 3 color samples red, green and blue. Since standard camera sensors can only detect one color value per pixel, an interpolation of the other values is needed to reconstruct the full color image. The task-data graph of the application, also

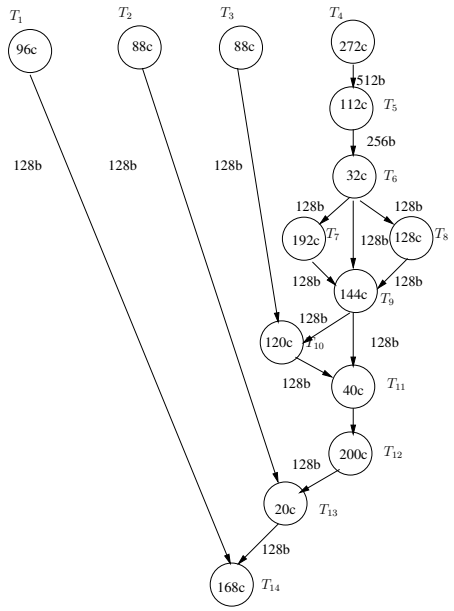


Figure 4. TMNR task graph, computation workloads are in cycles, and communication volumes in bytes.

used as benchmark in [BNB+09], has 12 tasks. The duration of tasks are extracted from profiling results by applying the application to 128×128 pixels image.

We experimented with the mapping of these applications on simplified models of the Spidergon architecture [CLM+04] with four or eight processors, using the five strategies described in Section III. We ran each of the five strategies with global timeouts of $10, 30, 60$, and 180 seconds. For each combination of global timeouts and strategy, we varied the per-query time bound as indicated in the following table.

global timeout (sec.)	per-query timeout (sec.)
10	$1, 10$
30	$5, 10, 30$
60	$5, 10, 60$
180	$5, 10, 180$

To evaluate the quality of the different methods we make use of a relative volume/area measurement [ZKT08]. For a particular problem instance, consisting of a task graph and an architecture, we take the union of the feasible costs found by any of the methods as a point of reference. Let us call this set R^*. The feasible cost space is trimmed to the smallest rectangle containing every point in R^* and the origin. We measure the area of forward closure of R^* in the trimmed cost space. Let us call this value a^*. For a given search strategy and timeout, consider the area of the forward closure of the feasible points found by this method and restricted to the trimmed cost space. Let us call this value a. The score of the method giving rise to a is $\frac{a}{a^*}$, which naturally tells us what portion the method contributed to the point of reference.

tasks	time	t/query	maxrect	union	bin	sat	rand
10	30	5	0.81	0.75	0.85	0.84	0.85
10	180	5	0.95	0.90	0.99	1.00	0.99
15	60	10	0.74	0.67	0.79	0.82	0.78
15	60	60	0.53	0.54	0.40	0.53	0.37
30	30	5	0.40	0.72	0.77	0.75	0.70
30	180	5	0.68	0.78	0.93	0.88	0.92
30	180	10	0.40	0.80	0.91	0.92	0.88
35	30	5	0.60	0.66	0.72	0.77	0.64
35	60	5	0.57	0.76	0.86	0.76	0.76
35	60	10	0.59	0.62	0.69	0.76	0.66
35	180	5	0.79	0.69	0.90	0.86	0.93
35	180	10	0.64	0.75	0.91	0.85	0.81
45	30	5	0.52	0.69	0.66	0.71	0.70
45	30	10	0.45	0.62	0.60	0.73	0.66
45	60	10	0.46	0.72	0.71	0.71	0.72
45	180	10	0.50	0.83	0.82	0.88	0.82

Table I
THE AREA PERCENTAGE FOR VARIOUS SYNTHETIC GRAPHS AND TIMEOUT CONFIGURATIONS ON SPIDERGON 8.

On small instances of graphs computational resources may be sufficient to obtain the real Pareto front in less than 180s. In that case the indicator used reflects the ability of each strategy to approach the optimum in a limited amount of time. As the size of the graph and architecture grow, unsatisfiability results become harder to prove, exceeding per-query time bounds. In these cases, the reference R^* may or may not coincide with the complete Pareto front and the indicator serves the comparison *between* strategies more than the global quality assessment.

For an oracle, we used Z3 [dMB08]. Our encoding of the communication cost and processor imbalance is exactly as described in Section II-A. The decision variables are mappings, and we use a Boolean variable for each task-processor combination to indicate when the task is mapped to the processor. The applications make use of the C API of Z3 and stores the logical context when possible between queries.

Tables I and II show the results per query. The under-approximation refinement strategies are labelled *maxrect* and *union* respectively. The distance reduction strategies are labelled *bin*, *sat*, and *rand*, representing respectively: binary search, search with satisfiable bias, and random search along the line segment as described in Figure 3. Amongst the five strategies, it appears the distance reduction strategy with a satisfiable bias (*sat*) performed the best. However, the presence or absence of a per-query time bound has a much *stronger* effect than the choice of strategy. The comparison between the 5 and 10 second time bounds is somewhat mixed, indicating a good choice for query timeout may be difficult to estimate in advance.

Although we do not present the data here, the methods

application	time	time/query	maxrect	union	bin	sat	rand
TMNR	30	5	0.80	0.75	0.83	0.86	0.84
TMNR	30	10	0.81	0.46	0.71	0.77	0.60
TMNR	60	5	0.87	0.78	0.83	0.91	0.88
TMNR	60	10	0.87	0.64	0.85	0.88	0.88
TMNR	180	10	0.95	0.92	0.95	0.96	0.94
Demosaic	30	5	0.76	0.66	0.29	0.79	0.52
Demosaic	30	10	0.62	0.45	0.29	0.51	0.52
Demosaic	60	5	0.79	0.79	0.74	0.77	0.77
Demosaic	60	10	0.82	0.69	0.72	0.76	0.64
Demosaic	180	10	0.86	0.83	0.73	0.76	0.84

Table II
THE AREA PERCENTAGE FOR TMNR AND DEMOSAIC APPLICATIONS AND DIFFERENT TIMEOUT CONFIGURATIONS ON SPIDERGON 8.

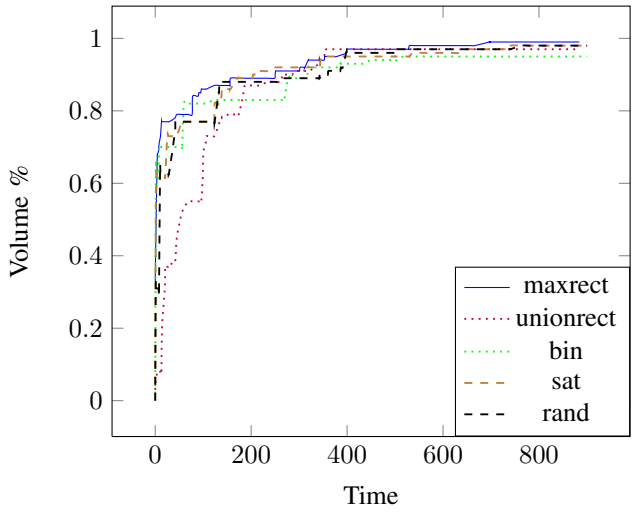


Figure 5. area percentage against time for TMNR on spidergon 8 with global timeout 900 and per query timeout 30

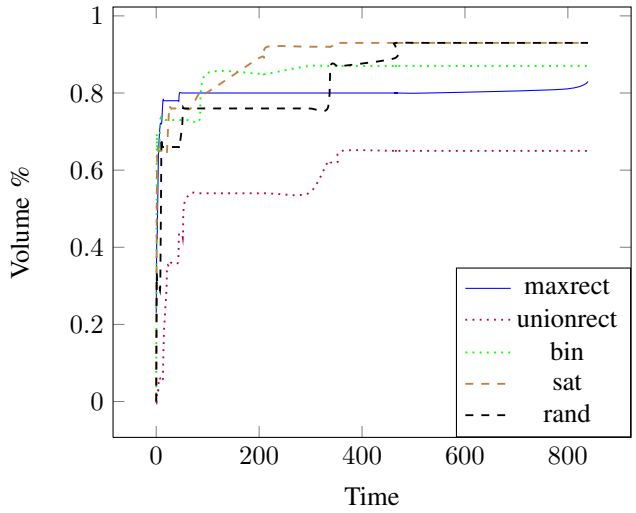


Figure 6. area percentage against time for TMNR on spidergon 8 with global timeout 900 and per query timeout 90.

performed much better on the Spidergon-4 than Spidergon-8. For example, in the 4 processor case, both industrial applications converged to the optimum within 180 seconds across all the strategies.

Figures 5,6 and 7,8 illustrate the evolution of the area percentage over execution time for both TMNR and Demosaic applications. The rate of improvement is quite high at the beginning and the different strategies converge rapidly to a reasonable approximation of the point of reference. However the rate then becomes much lower suggesting that this point is a good trade-off between precision and computation time on these problems. The *maxrect* strategy performs well in this setting. This is probably explained by the good locality between consecutive calls: the strategy only jumps from one part of the space to another when it can no more improve the current solution. This increases the benefit of saving the Z3 logical context between different queries. We conjecture that this contributes to the performance of the method.

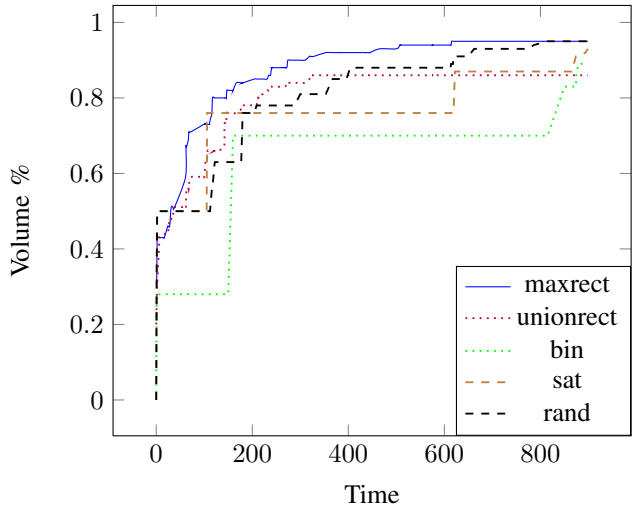


Figure 7. area percentage against time for Demosaic on spidergon 8 with global timeout 900 and per query timeout 30

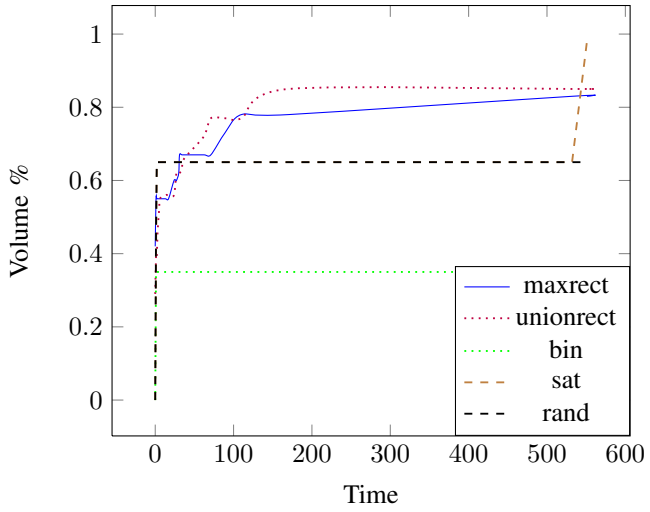


Figure 8. area percentage against time for Demosaic on spidergon 8 with global timeout 900 and per query timeout 900.

V. Conclusion and Future Work

We have conducted systematic experiments on the applicability of a modern constraint solver in finding approximate solutions for MCO problems arising from the deployment of data-intensive programs on multi-processor architectures. We have developed several methods for guiding the solver in choosing queries in the cost space. Due to the hardness of the constraint satisfaction problems and the fact that a solver may spend an enormous amount of time in trying to answer a query with a cost close to the optimum, we had to parameterize our search algorithms with a time budget per query so as to avoid getting stuck in dead ends. As it turns out, the significance of this parameter was larger than the choice of a particular search strategy. Unfortunately, it can be difficult to determine in advance a good value for the parameter and so perhaps applying the parameter dynamically in a slowly increasing fashion could be useful.

For the future we consider the following extensions of this work.

1) We intend to move to higher dimensions by considering additional cost criteria such as load balancing over the communication channels. Since the xSTream architecture is configurable and processors can operate in differing speeds or be shut down, we can also add the cost of the configuration (in terms of static power consumption) as an additional dimension in the spirit of the model introduced in [LM10];
2) Since the cost criteria chosen were approximative, ignoring possible congestions due to the precedence structure of the tasks, it will be interesting to compare the behavior of different mapping strategies using simulation on a model of the architecture. We are also working on applying the same methodology to the more difficult problem of scheduling, where a solution specifies not only *where* a computation or a communication takes place but also *when*. Some modeling problems related to the granularity of the communication should be resolved to this end;
3) Encouraged by some promising preliminary results, we are currently exploring new algorithms based on stochastic local search [HS04];
4) We intend to apply this methodology to perform *automatic data parallelization*. Video and radio applications admit uniform operations over large data blocks and the current practice of parallelizing their execution is by manual insertion of split/join operations. Since each choice of split/join boundaries induces a derived task-data graph, the existence of splitting strategies that admits a mapping of a given cost can be easily formulated as a query to the solver and benefit from the search strategies developed in this paper. Progress in this direction is a pre-condition for the development of high-level software engineering practices for parallel computing.

Acknowledgments: This work was partially supported by the French Minalogic project Athole.

References

[BBGM05] L. Benini, D. Bertozzi, A. Guerri, and M. Milano. Allocation and scheduling for MPSoCs via decomposition and no-good generation. *Principles and Practice of Constraint Programming-CP'05*, pages 107–121, 2005.

[BCA+09] J.J. Berenguer, N. Coste, I. De Poy Alonso, G. Desoli, E. Lantreibecq, J. Legriel, and G. Richard. xSTream: Architecture multi-coeur sur puce pour des applications multimedia. http://www.inrialpes.fr/vasy/multival/documents/xstream-2009.pdf, 2009.

[Ben96] A. Bender. MILP based task mapping for heterogeneous multiprocessor systems. In *Euro-DAC'96*, page 197, 1996.

[BFH+04] Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan. Brook for GPUs: stream computing on graphics hardware. In *SIGGRAPH '04*, pages 777–786. ACM, 2004.

[BNB+09] Sebastien Le Beux, Gabriela Nicolescu, Guy Bois, Youcef Bouchebaba, Michel Langevin, and Pierre Paulin. Optimizing configuration and application mapping for mpsoc architectures. In *Adaptive Hardware and Systems, NASA/ESA Conference on*, pages 474–481. IEEE Computer Society, 2009.

[CLM+04] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra. Spidergon: a novel on-chip communication network. In *System-on-Chip*, 2004.

[DdM06] B. Dutertre and L.M. de Moura. A fast linear-arithmetic solver for DPLL(T). In *CAV*, pages 81–94, 2006.

[Deb01] K. Deb. *Multi-objective optimization using evolutionary algorithms*. Wiley, 2001.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[dMB08] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *TACAS'08*, volume LNCS 4963, pages 337–340, 2008.

[DRW98] Robert P. Dick, David L. Rhodes, and Wayne Wolf. TGFF: Task graphs for free, 1998.

[EEP03] Cagkan Erbas, Selin C. Erbas, and Andy D. Pimentel. A multiobjective optimization model for exploring multiprocessor mappings of process networks. In *CODES+ISSS'03*, pages 182–187. ACM, 2003.

[Ehr05] M. Ehrgott. *Multicriteria optimization*. Springer Verlag, 2005.

[FLR00] C.P. Fenimore, J.M. Libert, and P. Roitman. Mosquito noise in MPEG-compressed video: test patterns and metrics. In *Proceedings of SPIE*, pages 604–612, 2000.

[HS04] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations & Application*. Morgan Kaufmann, September 2004.

[KTZ06] Simon Künzli, Lothar Thiele, and Eckart Zitzler. Multi-criteria decision making in embedded system design. In *System-on-Chip: Next Generation Electronics*, pages 3–28. 2006.

[LGCM10] Julien Legriel, Colas Le Guernic, Scott Cotton, and Oded Maler. Approximating the pareto front of multi-criteria optimization problems. In *TACAS*, pages 69–83, 2010.

[LM87] Edward A. Lee and David G. Messerschmitt. Synchronous data flow. In *Proceedings of the IEEE*, volume 75(9), pages 1235–1245, September 1987.

[LM10] J. Legriel and O. Maler. Meeting deadlines cheaply. Technical report, Verimag, 2010.

[LYH+08] W. Liu, M. Yuan, X. He, Z. Gu, and X. Liu. Efficient SAT-based mapping and scheduling of homogeneous synchronous dataflow graphs for throughput optimization. In *RTSS'08*, pages 492–504. IEEE, 2008.

[MA04] R.T. Marler and J.S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.

[RGB+06] M. Ruggiero, A. Guerri, D. Bertozzi, F. Poletti, and M. Milano. Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip. In *DATE'06*, pages 3–8, 2006.

[SLHT06] T. Schlichter, M. Lukasiewycz, C. Haubelt, and J. Teich. Improving system level design space exploration by incorporating sat-solvers into multi-objective evolutionary algorithms. In *Emerging VLSI Technologies and Architectures, 2006*, page 6. IEEE, 2006.

[TBHH07] Lothar Thiele, Iuliana Bacivarov, Wolfgang Haid, and Kai Huang. Mapping applications to tiled multiprocessor embedded systems. In *ACSD'07*, pages 29–40, Bratislava, Slovak Republic, 2007. IEEE Computer Society.

[TKA02] William Thies, Michal Karczmarek, and Saman P. Amarasinghe. Streamit: A language for streaming applications. In *CC '02*, pages 179–196, 2002.

[YHZ+09] Y. Yi, W. Han, X. Zhao, A.T. Erdogan, and T. Arslan. An ILP formulation for task mapping and scheduling on multi-core architectures. In *DATE'09.*, pages 33–38. IEEE, 2009.

[ZKT08] E. Zitzler, J. Knowles, and L. Thiele. Quality assessment of pareto set approximations. *Multiobjective Optimization*, pages 373–404, 2008.

[ZLT02] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In *Evolutionary Methods for Design, Optimisation, and Control*, pages 95–100, 2002.

[ZM02] L. Zhang and S. Malik. The quest for efficient boolean satisfiability solvers. In *Computer Aided Verification*, pages 641–653. Springer, 2002.

[ZT99] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. In *IEEE transactions on Evolutionary Computation*, volume 3(4), pages 257–271, 1999.