A Parameterized Compositional Multi-dimensional Multiple-choice Knapsack Heuristic for CMP Run-time Management *

Hamid Shojaei[1,2], AmirHossein Ghamarian[2], Twan Basten[2,3], Marc Geilen[2], Sander Stuijk[2], Rob Hoes[2]
[1] University of Wisconsin-Madison, Madison, WI
[2] Eindhoven University of Technology, Eindhoven, the Netherlands
[3] Embedded Systems Institute, Eindhoven, the Netherlands
shojaei@wisc.edu

ABSTRACT

Modern embedded systems typically contain chip-multiprocessors (CMPs) and support a variety of applications. Applications may run concurrently and can be started and stopped over time. Each application may typically have multiple feasible configurations, trading off quality aspects (energy consumption, audio-visual quality) with resource usage for various types of resources. Overall system quality needs to be guaranteed and optimized at all times. This leads to the need for a run-time management solution that selects an appropriate system configuration from all the application configurations of active applications. This run-time management problem can be phrased as a multi-dimensional multiple-choice knapsack (MMKP) problem. We present a compositional heuristic to solve MMKP, that due to the compositionality is better suited to CMP run-time management than existing heuristics that are all not compositional. Our heuristic outperforms the best-known heuristic to date. The heuristic is parameterized, leading to the additional advantage that it allows to trade off execution time vs. solution quality, and to bound the time needed to compute a solution. The latter makes it particularly well-suited for resource-constrained embedded platforms.

Categories and Subject Descriptors

C.3 [**Special-purpose and Application-based Systems**]: Realtime and embedded systemsC.5.4Computer System ImplementationVLSI Systems
General Terms: Algorithms, Design, Performance
Keywords: MMKP, CMP run-time management, Pareto algebra

1. INTRODUCTION

Complex dynamic applications are becoming increasingly dominant in different types of embedded systems, and ever more embedded systems have a chip-multiprocessor (CMP) as the compute platform. When (parts of) applications may start and stop over time, one of the most challenging issues is to select at run time a cost-effective mapping of the applications on these platforms.

Selecting an optimal configuration for each application requires optimization of power consumption, perceived quality, timeliness, and so on. This needs to be done within available platform resources and within performance constraints, and requires optimization of processing, memory, and communication resources of various types. This leads to a multi-dimensional design-space exploration problem. For dynamic systems with applications starting and stopping over time, this exploration needs to be done at run-time, within the resource-constrained embedded platform.

To alleviate run-time decision making, the available configurations of the various applications themselves may be obtained from an off-line Pareto analysis in the multi-dimensional quality/costs/resources trade-off space. The selection of a combination of the application configurations at run-time then requires composition of feasible Pareto-optimal configurations of applications into feasible Pareto-optimal configurations of a system, ultimately selecting one configuration per application, optimizing the overall cost and quality (in terms of e.g, power, perceived quality, etc.).

Figure 1 gives an illustrative example. Consider a system with three applications A, B, and C, that need to be mapped onto a systemwith 6 processing elements (PEs), allowing two clock speeds (ck1 and ck2 with ck1<ck2). Potential configurations are determined off-line. Some of the configurations are infeasible because of a latency constraint. The objective is to map applications onto the system so that latency and resource constraints are met and the energy consumption is minimal. Applications may start and stop at arbitrary times. When application A starts, a 4-PE configuration is selected with slow clock speed ck1. At time t1, application B starts and among all possible combinations, a configuration with fast clock ck2, 2 PEs assigned to each application, and energy consumption as low as possible is selected. This configuration is still valid when application C starts, which is assigned two additional PEs. Conceptually, in each step, the configurations of the newly arriving task are joined with the configurations of the already active tasks, making sure that clock speeds match (Join in the figure). Infeasible configurations, denoted in grey, and configurations that are guaranteed to be worse than others (dominated configurations) are removed (Const resp. Min in the figure).

In general, application configurations can be assumed to have a *value* (that takes into account all optimization objectives) and a *multi-dimensional resource usage*. We need

*This work was supported in part by the EC through FP7 IST project 216224, MNEMEE.

DAC'09, July 26–31, 2009, San Francisco, California, USA.

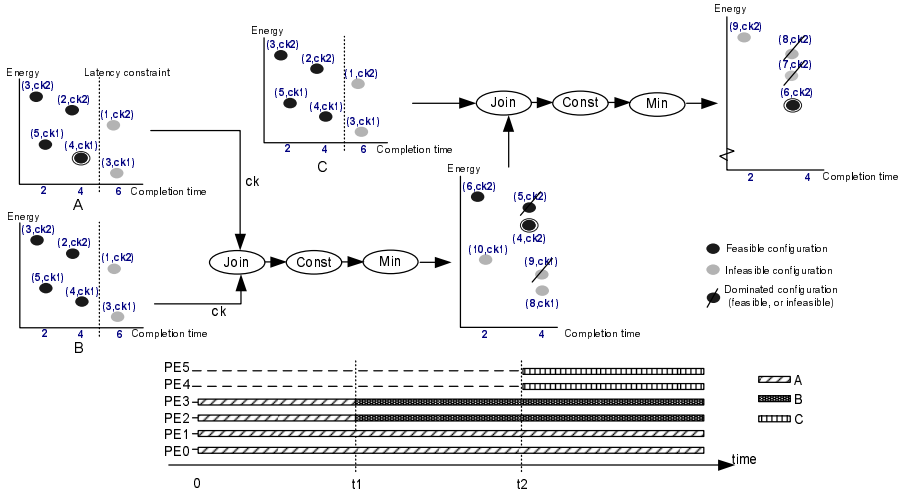

Figure 1: Configuration selection (adapted from [12]).

to choose exactly one feasible configuration per application, maximizing the total value of the selected configurations, without exceeding the resource constraints in any dimension. We do not explicitly consider other types of constraints (like the latency constraint in the example), because these can be treated in the same way as resource constraints. This leads to an optimization problem that is known as the Multi-dimensional Multiple-choice Knapsack Problem (MMKP, [8]). This optimization problem needs to be solved each time something changes in the set of active applications, i.e., each time an application starts or stops executing. We do not consider dynamic behavior within one application. This type of dynamic behavior can be handled independently, by other means like scenario-based design [3].

Let N be the number of applications, and N_i the number of configurations of application i ($0 \leq i < N$). Let R be the number of resources. The amount of available resources of type k ($0 \leq k < R$) is given by R_k. Suppose that configuration j of application i has value v_{ij} and requires resources r_{ijk} ($0 \leq k < R$). MMKP is then defined as follows, where the x_{ij} are binary-valued variables:

- *Maximize*
$$\sum_{0 \leq i < N} \sum_{0 \leq j < N_i} x_{ij} v_{ij}$$
- *Subject to*
  - $x_{ij} \in \{0,1\}, 0 \leq i < N, 0 \leq j < N_i$
  - $\sum_{0 \leq j < N_i} x_{ij} = 1, 0 \leq i < N$
  - $\sum_{0 \leq i < N} \sum_{0 \leq j < N_i} x_{ij} r_{ijk} \leq R_k, 0 \leq k < R.$

MMKP is an NP-hard problem. For CMP run-time management as sketched above, the MMKP problem needs to be solved at run-time, at a resource-constrained platform, each time the active application set changes. This puts severe constraints on the efficiency of the solution. There already exist various exact and heuristic algorithms for solving MMKP. We propose a compositional heuristic solution that follows the conceptual approach sketched in Figure 1. To the best of our knowledge, our solution is the first compositional solution to MMKP.

Our heuristic uses Pareto algebra [2], which is an algebraic framework for compositional calculation of Pareto-optimal solutions in multi-dimensional optimization problems. Pareto optimality is used as a central criterion to compare configurations and discard the ones that cannot be optimal. The practical complexity of the heuristic is determined by the number of configurations of the (sets of) applications considered in each optimization step. This provides a parameter to control the execution time of the heuristic. This parameter allows to trade off execution time and quality of the end result. It is furthermore possible to bound in practice the execution time of the heuristic for a given problem in terms of this parameter. This allows in turn to bound the amount of time allowed for the computation. This is particularly convenient in a context with hard or firm real-time constraints. The possibilities to trade off quality for execution time and to budget the execution time are unique to our heuristic.

The experimental results indicate that we achieve the same results in terms of speed and obtained value compared to the fastest non-compositional heuristic ([12]) known to date when assuming that all applications are started at a single point in time. If applications start and stop at different points in time, our heuristic shows execution times that are independent of the number of active tasks, in contrast to the execution times of non-compositional approaches that depend on the number of active tasks. The compositional approach is then generally much faster than the fastest heuristic so far.

The rest of the paper is organized as follows. The next section provides an overview of related work. Section 3 gives an introduction to Pareto algebra. Our Pareto-algebra-based heuristic is introduced in Section 4. Experimental results are presented in Section 5. Section 6 concludes.

2. RELATED WORK

Finding an optimal solution for an MMKP instance is not applicable for CMP run-time management, because the execution times increase dramatically with increasing problem sizes (see e.g. [4]). Also general heuristic search strategies such as genetic algorithms, applied to a knapsack variant in

[6], are not sufficiently fast.

The literature also describes some fast dedicated heuristic methods [1, 5, 8, 10]. Important ideas to improve efficiency originate from those papers, such as to consider the aggregate resource consumption of an application configuration, the idea to reduce the multi-dimensional search space into a two-dimensional search space, and the idea to reduce complexity by considering the configurations from each application that are on the convex hull. However, none of the heuristics is sufficiently fast for run-time management in a CMP context.

Researchers from IMEC propose in [12] a greedy approach to find near-optimal solutions for MMKP instances for CMP run-time management. This heuristic is by far the fastest among all heuristics to date. It is sufficiently fast for run-time management, finding a solution typically in the order of 1 ms, and providing results of good quality close to the best heuristics mentioned above. All (Pareto-optimal) application configurations are sorted according to value vs aggregate resource-usage ratio in a two-dimensional search space. Starting from the configurations with the lowest resource usage, a solution is constructed in a greedy way in a single traversal of this list by swapping configurations each time such a swap increases the value.

None of the mentioned approaches is compositional. In this paper, we propose a compositional approach building upon some of the ideas of [12] and earlier papers on MMKP. Besides the advantage of fast incremental decision making, our approach allows to trade off quality vs. execution time, and to bound the time spent on computing a solution.

3. PARETO ALGEBRA

This section provides an overview of Pareto algebra [2]. We briefly define the concepts and operations for multi-dimensional optimization in this algebra, to the extent that they are relevant for solving MMKP. We use the case study of Fig. 1 as a running example.

A *quantity* is a parameter, quality metric, or any other quantified aspect of a system. It is represented as a set Q with a partial order $\preceq_Q$ that denotes preferred values (lower values being preferred). The subscript is dropped when clear from the context. If $\preceq_Q$ is total, the quantity is *basic*; if $\preceq_Q$ is the identity, the quantity is *unordered*. In Fig. 1, *Energy* and *Completion time* are two basic quantities with total order $\leq$. Also the number of PEs in use is a basic quantity with order $\leq$. The quantity of clock speeds {ck1,ck2} is an unordered quantity, because there is no inherent preference for a low or high clock speed; the clock speed is a parameter of the system. Ultimately, the combination of clock speed and used number of PEs translates to energy usage and completion time, where lower values are preferred.

A configuration space $\mathcal{S}$ is the Cartesian product $Q_1 \times Q_2 \times \ldots \times Q_n$ of a finite number n of quantities and a configuration $\bar{c} = (c_1, c_2, \ldots, c_n)$ is an element of such a space.

A dominance relation $\preceq \subseteq \mathcal{S}^2$ defines configurations that are preferred over others. If $\bar{c}_1, \bar{c}_2 \in \mathcal{S}$, then $\bar{c}_1 \preceq \bar{c}_2$ iff for every quantity Q_k of $\mathcal{S}$, $\bar{c}_1(Q_k) \preceq_{Q_k} \bar{c}_2(Q_k)$. If $\bar{c}_1 \preceq \bar{c}_2$, then $\bar{c}_1$ *dominates* $\bar{c}_2$, expressing that $\bar{c}_1$ is in all aspects at least as good as $\bar{c}_2$. Dominance is reflexive, i.e., a configuration dominates itself. The irreflexive strict dominance relation is denoted $\prec$. A configuration is a *Pareto point* of a configuration set iff it is not strictly dominated by any other configuration. Configuration set $\mathcal{C}$ is *Pareto minimal* iff it contains only Pareto points, i.e., for any $\bar{c}_1, \bar{c}_2 \in \mathcal{C}$, $\bar{c}_1 \not\prec \bar{c}_2$. Fig. 1 shows the Pareto minimal configuration sets of the applications A, B, and C in our example.

A crucial observation is that, by allowing partially ordered quantities, quantities, configuration sets, and configuration spaces become essentially the same concepts. Pareto algebra exploits this to define operations on configuration sets that can be used to compute or reason about Pareto points for composite systems. We define those operations that are relevant for this paper. Let $\mathcal{C}$ and $\mathcal{C}_1$ be configuration sets of space $\mathcal{S}_1 = Q_1 \times Q_2 \times \ldots \times Q_m$ and $\mathcal{C}_2$ a configuration set of space $\mathcal{S}_2 = Q_{m+1} \times \ldots \times Q_{m+n}$. Let $\bar{c} \cdot \bar{d} = (c_1, \ldots, c_m, d_1, \ldots, d_n)$ for $\bar{c} = (c_1, \ldots, c_m) \in \mathcal{C}_1$ and $\bar{d} = (d_1, \ldots, d_n) \in \mathcal{C}_2$, and $k \in \{1, \ldots, m\}$.

- $\min(\mathcal{C}) \subseteq \mathcal{S}_1$ is the set of Pareto points of $\mathcal{C}$: $\min(\mathcal{C}) = \{\bar{c} \in \mathcal{C} \mid \neg(\exists \bar{c}' \in \mathcal{C} : \bar{c}' \prec \bar{c})\}$.
- $\mathcal{C}_1 \times \mathcal{C}_2 \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ is the (free) product of $\mathcal{C}_1$ and $\mathcal{C}_2$.
- $\mathcal{C} \cap \mathcal{C}_1 \subseteq \mathcal{S}_1$ is the $\mathcal{C}_1$-constraint of $\mathcal{C}$.
- Join($\mathcal{C}_1$,$\mathcal{C}_2$,Q)=$(\mathcal{C}_1 \times \mathcal{C}_2) \cap \mathcal{D}$, where $\mathcal{S}_1$ and $\mathcal{S}_2$ both include unordered quantity Q and constraint $\mathcal{D}$ is defined by $\mathcal{D} = \{\bar{c}_1 \cdot \bar{c}_2 | \bar{c}_1 \in \mathcal{S}_1, \bar{c}_2 \in \mathcal{S}_2, \bar{c}_1(Q) = \bar{c}_2(Q)\}$.

Fig. 1 illustrates the compositional computation of Pareto-optimal system configurations from Pareto-optimal configurations of the applications. The Join operation in each step enforces that the applications should run in configurations with the same clock speed; the constraint operation (Const in the figure; set intersection in Pareto Algebra) applies resource bounds for each dimension and removes infeasible configurations. The minimization operation removes dominated configurations. If we assume that the value of a configuration is inversely proportional to energy consumption, then the sketched (compositional) computation finds an optimal solution to the run-time configuration problem, maximizing value at all times. Note that the MMKP formulation assumes that all configurations of two different applications are always compatible, in contrast to the example where clock speeds need to match. An exact solution to MMKP can be formulated in terms of Pareto algebra in a straightforward way, following the example. Our MMKP heuristic given in the next section can easily be adapted to incorporate constraints like the need to match parameter settings such as the clock speed, making it only more efficient because less configurations need to be considered.

4. THE PARETO-ALGEBRA SOLUTION

This section introduces our Pareto-algebra-based solution for CMP run-time management, focusing on efficiently solving the underlying MMKP instances. It follows the approach of Figure 1. Each time the active application set changes, an MMKP instance needs to be solved. Consider the case that we have K active applications and M newly starting applications. The CMP run-time management approach sketched in [12] then takes an MMKP instance with $N = K + M$ applications. The MMKP instance that we need to solve takes $N = 1 + M$ applications, the M newly starting ones, and one 'application' that corresponds to the total set of active applications.

The $M+1$ configuration sets in our MMKP instance are all $R + 1$-dimensional, 1 dimension corresponding to the value and R dimensions corresponding to the resources. Conceptually, the solution takes the Cartesian product of these sets,

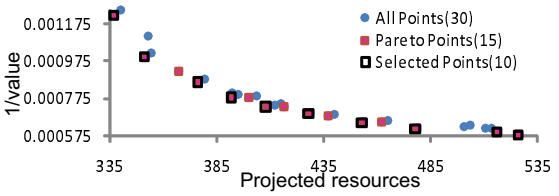

Figure 2: Minimization and reduction in a two-dimensional configuration space.

removes infeasible configurations, and removes dominated configurations. In this way, theoretically, all feasible configurations are computed, and a maximum-value configuration can be selected. In practice, this is intractable. To speed up the computation, we follow other MMKP heuristics by transforming all the configuration sets into 2-dimensional spaces. Dominance is checked in this 2-dimensional space. Furthermore, we avoid the explicit enumeration of the Cartesian product by interleaving the computation with the feasibility and dominance checks. Finally, we introduce a parameter L, which corresponds to the maximum number of Pareto-optimal configurations per application that we consider in each step. The projected 2-dimensional spaces are filtered to make sure that at most L configurations remain.

Given an MMKP instance I with N applications, it turns out the execution time $T(I)$ of our heuristic is bounded in practice as follows:

$$T(I) \leq C \cdot N \cdot L^2, \tag{1}$$

where C is some constant depending on the processor used. From this bound, it follows that we are able to dynamically at run-time bound the time budget we want to spend on solving an MMKP instance. Furthermore, lower values for L, meaning that less Pareto-optimal configurations are kept in each step of the heuristic, typically result in lower quality of the final solution found. This allows (dynamic) trade offs between quality and execution time.

Before presenting the algorithm in detail, we first discuss the projection of a configuration set into a 2-dimensional space, and the selection of L points from such a space. Fig. 2 gives an example from one of the benchmarks, I6, introduced and studied in detail in the next section.

One dimension in the 2-dimensional space is the value dimension, where we take the inverse value for illustration purposes, so that less is better in the figure. The second dimension is a projected resource usage. Ref. [1] suggests the transformation of the R resource dimensions to a single dimension by taking the distance to the origin in the resource subspace (that omits value). To avoid expensive square and square-root computations, we simply take the sum of all resources: if $\bar{r}$ is a configuration $(r_0, \ldots, r_{R-1})$ in the resource subspace, then the projected resource value $r^*(\bar{r})$ is defined as $\sum_{0 \leq k < R} r_k$. In Fig. 2, the blue points represent the resulting configurations in the derived 2-dimensional space.

In Fig. 2, the blue points represent the resulting configurations in the derived two-dimensional space.

As mentioned, Pareto points are determined in this 2-dimensional space. The red configurations in Fig. 2 are the Pareto points in the example.

It remains to filter out L Pareto points from this space, where L is assumed to be at least 2. Suppose points in the 2-dimensional space are represented in terms of variables v (value) and r (projected resource). For any given constant c, $v \cdot r = c$ then provides a line through the origin of the space. We select our L points, by first taking the 2 Pareto points $(r_{\max}, v_{\max})$ and $(r_{\min}, v_{\min})$ that result in two lines $v_{\max} \cdot r_{\max} = c_{\max}$ and $v_{\min} \cdot r_{\min} = c_{\min}$ with maximal and minimal values for the constant c, respectively. We then divide the range $c_{\max}..c_{\min}$ in $L-1$ equally large parts, resulting in $L-2$ new lines through the origin of the graph. For each of these lines, we take a Pareto point closest (Euclidean distance) to the line. In this way, we select the extreme Pareto points in the space, plus a good spread of Pareto points over the remainder of the space. This provides the highest possible flexibility in selecting fitting configurations for an application.

Algorithm 1 Pareto-Algebra heuristic for MMKP

```
1: MMKP(S, Rb, L)
2: for all C_i ∈ S do
3:    min(C_i);
4: end for
5: lp = Convert_MMKP_to_lp
6: vb = lp_solve(lp);
7: append vb to Rb;
8: ConfSet Cm = S[0];
9: S = S − S[0];
10: for all C_i ∈ S do
11:    if size(Cm) > L then
12:       reduce(Cm, L);
13:    end if
14:    if size(C_i) > L then
15:       reduce(C_i, L);
16:    end if
17:    Cm = SumBoundMin(Cm, C_i, Rb);
18: end for
19: return maxVal(Cm);
    SumBoundMin(Cm, C, Rb)
20: ConfSet res = empty;
21: for all c_i ∈ Cm do
22:    for all c_j ∈ C do
23:       if Constrain(c_i, c_j, Rb) then
24:          AddAndMin(res, product(c_i, c_j));
25:       end if
26:    end for
27: end for
28: return res;
```

Algorithm 1 provides pseudo-code for our MMKP heuristic. The algorithm gets as input an N-sized vector of configuration sets (S), an R-sized vector of resource bounds (Rb), and the value for our parameter L. The first step of the algorithm uses the *min* operation to find Pareto-optimal configurations of each application (Lines 2, 3). The algorithm then transforms the MMKP problem to linear programming constraints and uses *lp_solve* [7] to compute an upper bound for the objective function (value, Lines 5, 6). This upper bound is added to the resource bounds to consider as a bound for the value dimension in the *Constrain* function (Line 7).

The compositional part of the algorithm is implemented from lines 8 to 18. The for-loop in Line 10 iterates over applications and in each step configurations of the new application are combined with already found configurations. In each iteration, the algorithm first limits the size of configuration sets C_i and Cm to L, following the procedure given above. We then use the *SumBoundMin* function for combining configurations. This function gets two configuration sets and a multi-dimensional resource bound. It computes the feasible, Pareto-optimal configurations in the Cartesian product of the two configuration sets. In each step of this

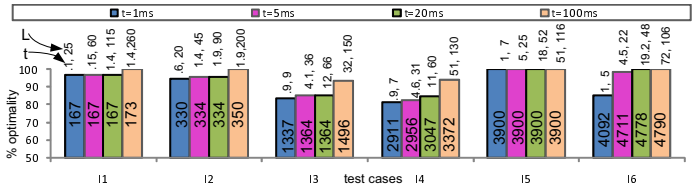

Figure 3: Controlling time budget.

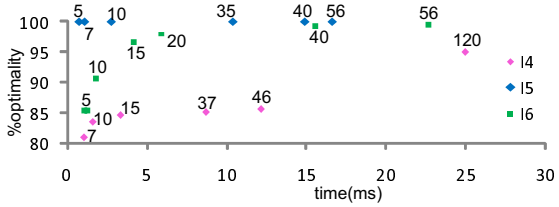

Figure 4: Trading off value vs. execution time.

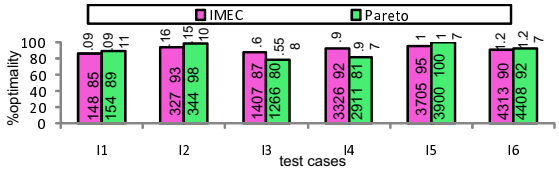

Figure 5: Comparison with the IMEC heuristic.

computation, the function sums all corresponding dimensions from two considered configurations elementwise, and it keeps only the resulting configurations that meet the *multi-dimensional* resource bound and are at that moment in time Pareto-optimal (Lines 23, 24).

After all compositional steps over the applications, Cm contains all feasible configurations of the system that are Pareto-optimal in the projected 2-dimensional space. The algorithm then returns the configuration with the maximum value among those configurations (Line 19).

It is interesting to consider the theoretical worst-case complexity of the algorithm. The complexity of the first part of the algorithm up to Line 8 is determined by the linear programming call of Line 6. The complexity in each step of the compositional part depends on the number of configurations in the considered configuration sets (Cm and C_i), and on parameter L. In each step of the optimization, the algorithm first limits the size of configuration sets C_i and Cm to L; the second part invokes SumBoundMin on the reduced Cm and C. The two size-limiting phases each require a sorting of the configuration list and a walk through the input configuration list to pick L configurations. Therefore, observing that the size of any of the intermediate Cm results is bounded by L^2 and assuming $N_{\max}$ is the maximum number of configurations in any of the input sets in S, then the complexity of these two reductions is $O(L^2 \log L)$ and $O(N_{\max} \log N_{\max})$. In the second part, the complexity of SumBoundMin is $O(L^4)$, as it has two nested loops of size at most L and the statement of Line 24 has complexity $O(L^2)$. Since $N-1$ compositional steps are made by Algorithm 1, the overall complexity of the compositional part of the algorithm, which is the part that is executed online, becomes $O(N(\max(N_{\max} \log N_{\max}, L^4)))$. In practice, the running time of the algorithm typically never shows its worst-case behavior, because the Pareto-dominance checks and the checks on the resource and value bounds prune the total search space, and because $N_{\max}$ is often (much) smaller than L. In the experiments in the next section, the execution-time bound of Eq. (1) that is quadratic in L is always conservative. If necessary, the bound can be tuned to specific cases.

In any case, by increasing the value of L we can change the behavior of the algorithm from a very short running time and solutions with potentially limited quality towards longer running times and solutions close to the optimum solution of the MMKP instances. By storing Pareto-optimal feasible configurations of the active applications in a system, the complexity of the sketched CMP run-time management approach building upon the heuristic becomes independent of the number of active applications, and it depends only on the number of new applications.

5. EXPERIMENTAL EVALUATION

In this section, we experimentally investigate the various aspects of our approach, and compare our heuristic to the fastest state-of-the-art heuristic of [12], which is the only one sufficiently fast for CMP run-time management. We implemented this heuristic and our compositional heuristic, using the C programming language. We tested the heuristics on the cycle-accurate SimIt-ARM simulator [11] for the StrongARM architecture running at 206 MHz. The results we achieve for the IMEC heuristic [12] are in line with the results reported in [12] for the same platform.

Experiments are reported for the standard MMKP benchmarks available via [9]. The thirteen benchmarks, I1-I13, have values for N (number of applications), N_i (number of configurations per application i), and R (number of resource dimensions) up to 400, 10, and 10, respectively. In line with [12], we divide the benchmark problems in two groups. Instances I1-6 are representative for CMP problem sizes with $N \leq 30$, $N_i \leq 10$, and $R \leq 10$. The second group I7-I13 provides larger test cases. We focus our experiments on the 1-I6 benchmarks, and only briefly investigate the scalability of our approach to larger problem instances.

An important strength of our heuristic is the possibility to bound the time needed to compute a solution, via Equation (1). In the first set of experiments, we bounded the computation time to 1, 5, 20, and 100 ms. To compute L from a time budget, we use Eq. (1) with C equal to 0.0003 for the considered StrongArm configuration. Fig. 3 illustrates for the CMP problem sizes the value of parameter L, the real execution time t, and the quality of the solution in each case (given both as a bar on a normalized scale with the optimum value as a reference, and as an absolute number in that bar). For some problems, the size of intermediate configuration sets are never close to the computed L. Consequently, the real execution time is far below the considered bound. All the results confirm that the bound of Eq. (1) is

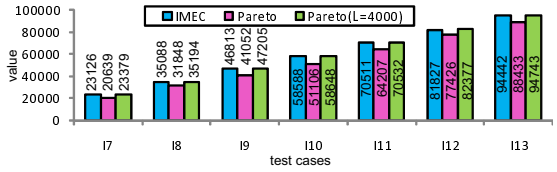

Figure 6: Large MMKP instances

conservative.

Fig. 4 illustrates the trade-off between quality of the solution and the computation time for our heuristic. The results are reported for the three largest CMP benchmarks, I4-I6, with different values for L, given as annotations in the figure. The results indicate that the trade-off can be controlled by L. For I5, the resource bounds are very tight and the number of feasible configurations after each step of the compositional computation remains very low. So in this case, we can have an exact solution with a small L. The resource bounds in I4 are more relaxed and we should consider larger values for L to find a good-quality solution.

An important issue is how our heuristic compares to the IMEC heuristic for the same problem. Fig. 5 shows execution times for the CMP benchmark in the worst-case situation for our approach in which all applications start simultaneously without any active applications. For this experiment, we considered the execution time needed by the IMEC heuristic as a bound for the computation time for our heuristic, deriving L from this bound. In Fig. 5, the numbers on the bars indicate the measured computation time and, for our heuristic also L. The execution times for our heuristic are always within the bound determined by the IMEC heuristic. The numbers inside the bars give obtained absolute and relative (%) values. On average, we may conclude that our obtained solution quality is as good as that of the IMEC heuristic.

It is also interesting to consider the large test cases, despite the fact that these are not the target for our heuristic. Fig. 6 shows experiments comparing the solution quality of our heuristic to the solution quality of the IMEC heuristic. When taking the time needed by the IMEC heuristic on a 2 GHz laptop with 2 GB main memory and running Windows XP, as the budget for our heuristic, we obtain $L = 10$ in all cases. For those cases, we obtain a lower quality than IMEC, illustrating that our heuristic scales worse in terms of quality given a fixed time budget. When increasing the value of L, however, we can obtain better quality in all test cases, at the expense of extra execution time. For these large cases, the projection of resources into one dimension leads to poor results, so Pareto points are computed in the multi-dimensional space.

Finally, we evaluate the impact of compositionality, a second important strength of our heuristic. Fig. (7) compares the performance of the compositional heuristic with the IMEC heuristic, for benchmark I5 with $L = 10$ and assuming that applications start in groups of 5 or 1. The results confirm that the performance of our heuristic does not depend on the number of active applications. The IMEC heuristic needs to solve the complete MMKP instance for all applications each time, which results in increasing execution times when more applications are active. It is clear that our heuristic outperforms the IMEC heuristic when applications start at different points in time.

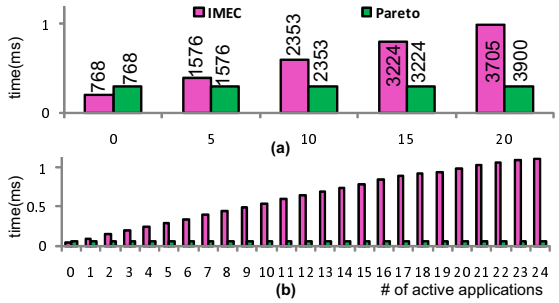

Figure 7: Compositionality

6. CONCLUSION

We have presented a novel parameterized compositional heuristic to solve MMKP. This heuristic may form the basis of a run-time quality and resource manager for CMPs. It allows to select application configurations at run-time, in such a way that overall system quality is optimized. The heuristic outperforms the best known heuristic to date (that is not compositional). The parametrization furthermore allows to trade off the time needed to find a solution for solution quality, and to bound the time used for finding a solution. These properties imply that it fits well with the constraints imposed by embedded applications and resource-constrained embedded platforms. In future work, we plan to consider the actual reconfiguration of a CMP when new applications enter or leave the system, ultimately aiming at the development of a prototype run-time quality and resource manager for CMPs.

7. REFERENCES

[1] M. Akbar et al. Solving the multidimensional multiple-choice knapsack problem by constructing convex hulls. *Computers and Operations Research*, 33(5):1259-1273, 2006.

[2] M. Geilen et al. An algebra of Pareto points. *Fundamenta Informaticae*, 78(1):35–74, 2007.

[3] S.V. Gheorghita et al. Application Scenarios in Streaming-Oriented Embedded-System Design. *IEEE Design and Test of Computers*, 25(6):581-589, 2008.

[4] S. Khan et al. The utility model for adaptive multimedia systems. *Int. Workshop on Multimedia Modeling*, p. 111–126, 1997.

[5] S. Khan et al. Solving the knapsack problem for adaptive multimedia systems. *Studia Informatica Universalis*, p. 161–182, 2002.

[6] S. Khuri et al. The zero/one multiple knapsack problem and genetic algorithms. *ACM Symp. of applied computation*, 1994.

[7] Lp_solve. http://lpsolve.sourceforge.net.

[8] M. Moser et al. An algorithm for the multidimensional multiple-choice knapsack problem. *IEICE Trans. on Fundamentals of Electronics*, E80-A(3):582-589, 1997.

[9] MMKP Problems. ftp://cermsem.univparisl.fr/pub/CERMSEM/hifi /MMKP/MMKP.html.

[10] R. Parra-Hemandez and N. Dimopoulos. A new heuristic for solving the multichoice multidimensional knapsack problem. *IEEE Trans. on Systems, Man, Cybernetics*, 35(5):708-717, 2005.

[11] SimIt-ARM. http://simit-arm.sourceforge.net.

[12] Ch. Ykman-Couvreur et al. Fast Multi-Dimension Multi-Choice Knapsack Heuristic for MP-SoC Run-Time Management. In *SoC 2006*, p. 1–4. IEEE, 2006.