Using Constraint Logic Programming in Memory Synthesis for General Purpose Computers

Renate Beckmann* and Jürgen Herrmann**
University of Dortmund
Department of Computer Science XII* and I**
beckmann@ls12.informatik.uni-dortmund.de
D-44225 Dortmund

Abstract

In modern computer systems the performance is dominated by the memory performance. Currently, there is neither a systematic design methodology nor a tool for the design of memory systems for general purpose computers.We present a first approach to CAD support for this crucial subtask of system level design. Dependencies between influencing factors and design decisions are explicitly represented by constraints and constraint logic programming is used to make the desing decisions.

The memory design is optimized with respect to several objectives by iterating the (re)design cycle. Event driven simulation is used for evaluation of the intermediate results. The system is organized as an interactive design assistant.

1. Motivation

During the recent years, the complexity and capabilities of microelectronic systems has grown significantly. As a consequence, the design of these systems has also become more complex and time-consuming. Therefore, a powerful tool support is indispensable for the design of complex microelectronic systems. During the course of time, design automation tools became available at higher and higher levels of abstraction. Layout editors, to a large extent, have been replaced by placement and routing tools. These have been complemented by logic synthesis. Logic synthesis, in turn, is expected to be complemented by high-level synthesis. As evidenced by recent commercial announcements, high-level synthesis is currently made commercially available.

One crucial issue of system level design is memory synthesis. It is nowadays widely accepted, that the performance of the memory system for a general purpose computer such as a workstation or PC dominates the performance of the computer as a whole [Wulf95]. There is a serious concern that the speed of memory systems will continue to match the speed of processors [Wilk95]. Typically, memory access slows down the execution speed for processor instructions significantly. Due to the increasing speed of processors, these memory systems have become more and more complex to supply the required bandwidth. Sophisticated techniques such as hierarchical organization of memory components, complex caches, interleaving, pipelining, bus snooping etc., which previously have only been available on expensive mainframes, have found their application in mass products. For multiprocessor systems, the design of powerful memory systems is even more complex.

The synthesis of memory systems is characterized by numerous influencing factors and design decisions to be made. There are many complex dependencies between theses factors and decisions that must be considered, many one of them being vague, heuristic or unknown. Therefore, it is hard to get a clear picture of all relations between influencing factors and design decisions for memory design. Moreover, memory design is a multidimensional optimization problem, i.e., there are several objectives to be considered. Besides memory performance (average access time, miss ratio, etc.), for instance the cost of the required off-chip memory components and the area consumption of on-chip caches have to be taken into consideration.

Despite the crucial importance of memory design for general purpose computers, there is no systematic design methodology or theory for this complex task. Consequently, the design of such memory systems is governed by rules of thumb. These vague heuristics reflect the knowledge of “experts” more or less familiar with this area. Simulation is used to validate some of the design decisions, but the current situation in processor memory design can be described by the following statements:

- Processor memory synthesis for general purpose processors is currently more an art than a science, i.e., it is not an engineering discipline.
- Design decisions are mostly based on the mentioned rules of thumb and sometimes time-consuming analyses of their consequences (see below).
- There are no CAD tools supporting memory synthesis for general purpose computers.

As a result, even major industrial companies are sometimes surprised by the (lack of) performance of the memory systems.[1]

In this paper, a first approach to CAD support of memory synthesis for general purpose computers is presented. With this approach, a systematic procedure of memory synthesis shall be supported. The consideration and comparison of several design alternatives is enabled.

Our approach features an explicit representation of the dependencies between influencing factors and decisions for memory synthesis. The representation is based on constraint logic programming. The implemented prototype creates an initial memory design that is redesigned and optimized stepwise according to the objectives. The analysis of each intermediate design result is performed by an event-driven simulation. Design decisions can be influenced and revised by the user, i.e., an interactive design style is supported by the system.

The rest of the paper is organized in the following way: Section 2 presents current work on memory synthesis. The memory synthesis task and the main features of this task, relevant for the organization of our system, are discussed in Section 3. Section 4 gives a brief introduction into constraint logic programming. The conception of the new system for memory synthesis SPEISE is presented in Section 6. Results of the implemented prototype and topics for further research are documented in Section 7. Section 8 concludes the paper.

2. Current Work

Until now, only very limited work on CAD for memory synthesis has been published. Most of these approaches are on a lower level of abstraction than the memory synthesis problem addressed in this paper. The available papers either deal with ASICs, with a specific, limited class of processors or are restricted to small subtasks of memory synthesis:

- *Memory Synthesis for DSP applications*
  PHIDEO [vMee92] is a silicon compiler for digital signal processors. During memory synthesis, instances of the appropriate memory types are allocated and the addressing mechanisms are selected. The allocated simple memory modules are used to delay the digital signals according to the timing constraints.
- *Register allocation*
  Register allocation is performed in compilers [Aho86] and high-level synthesis tools [McFa90; Gajs92].
- *Allocation of multi-port memories in high-level synthesis*
  Previously allocated, isolated registers are merged into multi-port memories [Bala88].
- *Buffer allocation in high-level synthesis tools*
  In [Kolk93] a method is presented for minimizing sizing of communication buffers in an environment of communicating concurrent processes.

For these specific subtasks of memory synthesis, methods and algorithms are available. For instance the number of allocated registers can be calculated from the maximum number of variables referenced concurrently during a singe control step. On the other hand, there is no CAD tool supporting the memory synthesis problem for general purpose computers, described above. Instead, there are several publications about the influence of single memory parameters on the performance of the memory system as a whole. Most of the analyses are related to cache parameters. Some examples of these analyses are listed below:

- In [Smit82; Henn96; Przy90] the effect of the selected prefetch strategy, the selected strategy for updating the main memory, the line size, the number of sets and several other design decisions on the miss ratio is considered.
- In [Kris96] performance modelling for computer architecture is described. Analytical models representing the effects of cache design decisions on the performance are described in [Agar89; Berg93; Kris96; Saav95]. These models cannot be used for memory synthesis, because they deal only with few design decisions. Models cannot be used to analytically predict the performance of a designed memory system. Therefore simulation is necessary to examine the effect of a design decision on the performance of the whole memory system.
- In [Rau91] the effect of the input buffer size on the performance of interleaved memories is analyzed.

Depending on these analyses some exact resp. heuristic dependencies between the input data and the different design decisions can be derived. Typically, heuristic dependencies are formulated as qualitative relations. To be used in a memory synthesis system,

1. This observation can be made for cache design of modern SPARC systems such as the SPARC-10. According to our knowledge, cache block sizes and the interleaving factor are not well balanced.

they have to be quantified. Besides, information about many crucial design decisions is still lacking. Moreover, there is no published work for a comprehensive treatment of the numerous influencing factors and design decisions for memory synthesis.

3. Memory Synthesis

In the following we start with a characterization of the memory synthesis task and then analyze the task features.

3.1 The Task

As mentioned above, the CPU performance is currently improved at a much faster rate than that of memories. To manage the resulting problem, a sophisticated memory organization must be designed. A memory system nowadays does not consist of a single component but of a hierarchy of memory components ranging from small, fast and expensive ones, placed near the CPU (i.e., register, buffer, first level cache), to large, slower and cheaper ones (i.e., second level cache, main memory, secondary memory). Registers are allocated

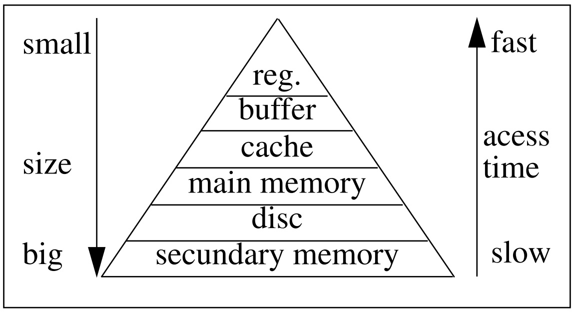

Fig. 1: memory hierarchy

for example, during high-level data path synthesis. In memory synthesis for general purpose processors the main component under design is the cache because it is large enough to hold a moderate amount of data, more than a register. And furthermore an access to a cache is fast enough for processor access because it is now possible to place the cache on the processor chip.

To speed up a memory reference, data requested by the CPU, has to be available in a fast memory component or the requests must be parallelized. In either case this requires a good organization of the memory system. Design decisions are e.g., number of cache levels, size, associativity, and prefetch strategy of a cache, size and degree of interleaving of the main memory, etc.

Which memory system is a “good” one, depends on time and order of the different data requests. These, in turn, depend on two factors: application programs and underlying computer architecture. Information about the characteristics of application programs (i.e., time and locality of the referenced memory addresses and working set) is necessary. For instance, if the application programs consists of many loops accessing the same data in each pass, the amount of different data that is accessed is small. In this case the working set of the application program is also small and the locality is high. This may lead to a small cache. The exact order and time of the memory references is also effected by the underlying computer architecture. Therefore the memory system under design has to be suitable for the given computer architecture. For instance, in RISC architectures with pipelining, memory accesses to instructions and data can be parallelized by designing separate caches for instructions and data. Another relevant feature is the cycle time of the processor. This is an upper bound for the access time of a first level cache.

The main objective in memory synthesis is to configure a memory system that minimizes the access time of memory references. But there are some other criteria like chip area and cost that restrict the design space of a memory system.

The memory synthesis task described above can be summarized as follows:
For a given general purpose computer architecture and a class of application programs a memory system has to be designed and optimized according to given criteria like access time, chip area and cost.

3.2 Problem Features

The structure of memory synthesis task is analyzed in order to derive design decisions for a memory synthesis tool.

- Memory synthesis for general purpose processors is a *new area of research*. As mentioned above there exists no tool support on this *high level of abstraction*. Currently memory synthesis is supported by tools only for ASICs. For general purpose computer systems only analyses exist. These analyses have to be combined to get information required for building a memory synthesis tool. To collect some experience in modelling and synthesis techniques as fast as possible, it is very helpful to build a p*rototype*. *Logic programming* is an adequate paradigm, because it supports an abstract level of programming that speeds up the programming process rapidly.

- Memory synthesis is a *complex synthesis task* (see above).
- The memory synthesis task consists of s*ome structuring and a lot of dimensioning decisions*. The structuring decisions select the components of the memory configuration. The number of different structural decisions is small, because there exist only few principally different memory configurations. The dimensioning decisions determine the features of each memory component (i.e., size, block size, replacement strategy). The number of dimensioning decisions is limited but large.
- The *design decisions are of different types*: structural, numerical, boolean or symbolical (for details see Subsection 6.3.).
- The numerous existing analyses of single design decisions show that memory synthesis is dominated by a *huge amount of relationships* between different design decisions and the features of the design environment (computer architecture, application programs, and objectives). These relations can be expressed by *constraints*. A programming paradigm supporting prototype development and constraints is *constraint logic programming* (see Section 5.).
- Up to now most of the *relations* described above, especially those between the design decisions and the objectives, are expressed *qualitatively.* For memory synthesis these relations must be *quantified* by use of *heuristics*. Additionally most of these relations are *not monotone*. Increasing the value of some design decision raises the performance only to a certain extent. For instance, increasing the cache size decreases the miss ratio. To a certain extent the average access time is decreased because less data has to be fetched from main memory. But the larger the cache the slower a single access to the cache and the slower the average access time. A synthesis tool must be able to model these dependencies.
- The memory synthesis task is a *multidimensional optimization problem* (see above).
- Up to now *no objective function* is known that quantifies the relations between the design decision and the objectives in form of a formula. The only way to measure the performance of the memory system under design due to the underlying computer architecture and the application programs is simulation. Memory references to the synthesized memory system have to be simulated.
- Due to the lack of a *formal description and theory for memory synthesis,* (no quantitative relations and no objective function, and no compound method for considering different objectives adequately and concurrently) a stepwise optimization is necessary by using *redesign cycles.*

4. Constraint Logic Programming (CLP)

In this section we describe why we use constraint logic programming for memory synthesis.

4.1 Basic Idea

Constraint logic programming extends logic programming by a mechanism for constraints modelling and processing [Frue93]. Constraints express relations between technical parameters of the problem. The idea of CLP is to restrict the search space, as much as possible, by constraints and to search the remaining space in a moderate amount of time. The processes of constraint handling and search are intertwined. Each constraint is imposed but the execution is delayed until the constraint can be evaluated without anticipating any search decisions. When during the searching phase some technical parameters are restricted the relevant constraints are resumed and executed. Additionally this search can be done in a heuristic, problem specific way.

In memory synthesis the relations between design decisions and environment features are expressed as constraints. The process of making design decisions, called labelling, is done heuristically.

Restricting the search space before and during labelling, improves the solution process drastically. In logic programming without constraints the design decisions are selected in the unrestricted decision space. This may cause a lot of wrong decisions and implies a large amount of backtracking, slowing down the solution process. Constraint logic programming can avoid most of these wrong decisions and the resulting backtracking.

4.2 Memory Synthesis as CLP Problem

In memory synthesis as described above, a memory system can be represented by a generic model. Each design decision is represented by a parameter with a domain representing the alternatives of this decision. For instance, for each cache (data cache, instruction cache, or second level cache) there are parameters for size, block size, associativity, replacement strategy, write strategy, etc. Restricting the search space is done by restricting the domains of some parameters. Decision making corresponds to instantiating parameters.

The design decisions are of different types: Structural ones determine the components of the memory configuration. For each component there are several dimensioning decisions of boolean, numerical or symbolical type. As denoted above, all decisions are represented by parameters with different domains. The types of these domains range from numerical (i.e., size), over symbolical (i.e., replacement strategy), to boolean (i.e., on-chip integration). These types are also used for the domains of structural design decisions (i.e., availability of a second level cache is a structural design decision represented by a boolean parameter). So for the whole synthesis task it is possible to use a homogeneous search strategy that can be handled well by a constraint system.

Architectural features and application features can be represented as generic models, too. The architectural parameter values are given by the designer and the application ones can be extracted by analyzing memory reference sequences of application programs (see below).

Memory synthesis, described here, differs from the standard CLP problem in two ways: Firstly the set of constraints, extracted from the analyses described in literature, may be inconsistent. For this reason each constraint is extended by a weight expressing the importance of that constraint. If an inconsistency occurs, constraints with small weights are relaxed successively. The sum of the weights of the remaining consistent constraints should be maximized. Secondly the objectives cannot be calculated by a formula. They have to be calculated by simulation of given memory reference sequences on the synthesized memory configuration.

Definition. Let V be a set of variables $\{v_1,...,v_n\}$ representing architectural, application, and memory parameters, each v_i with a domain D_i of possible values. Let C be a set of constraints $\{c_1,...,c_m\}$ expressing the relations between variables in V: $c_j \subseteq D_1 \times ... \times D_n$. A weight function $w_c(c_j)$ gives the weight of each constraint denoting the importance. Let O be a set of objectives $\{o_1,...,o_r\}$ calculated by simulation of traces on the memory configuration and $w_o(o_k)$ a weight function expressing the user given priorities to each of the objectives o_k. A function $eval(o_k)$ evaluates the quality of the design with respect to the objective o_k. $(x_1,...,x_n)$ is an optimal solution of the problem if the following conditions hold:

- $\forall i \quad x_i \in D_i$
- $\sum_{c_j \in C'} w_c(c_j) \rightarrow max$
- $\sum_{k=1}^{r} w_o(o_k) \cdot eval(o_k) \rightarrow max$

with $C' = \{c \in C | (x_1, ..., x_n) \in c\} \subseteq$
and $<o_1, ..., o_r> = simulation(x_1, ..., x_n)$.

Each parameter x_j of a solution is instantiated to a value of D_j. The weighted sum of constraints consistent with the solution is maximized. And the weighted sum of objective evaluations is maximized.

A small example shows the power of constraints in the domain of memory synthesis: For a simplified cache synthesis the cache hierarchy consists of 1 or 2 cache levels where the first level cache may be split into one for instructions and one for data. For each cache 13 design decisions with varying domain sizes have to be made. The decision space has a size of $0.5 * 10^{20}$. After imposing 20 of 50 constraints the design space is restricted to 10^7. After determining 2 of the design decisions (size and block size of each cache) the remaining 30 constraints restrict the search space to 430 design possibilities, which are examined heuristically (Fig. 2).

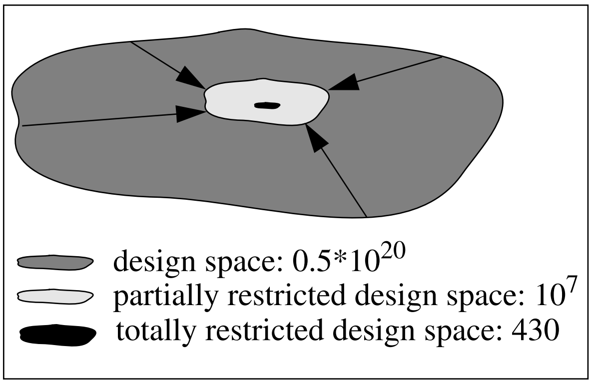

Fig. 2: design space restriction

Tab. 1 gives a concrete example that again illustrates the power of search space restriction.
The domains of six parameters are shown before and after restriction by the given constraints. The search space size (multiplication of the parameter domain sizes) is reduced from about $6*10^6$ to 18.

parameter	before	after
working set locality	small big	small big
size (KB) line size (B) associativity replacement	8 .. 256 4 .. 256 1 .. 32, full no, rnd, lru	8, 16, 32 16, 32 2, 4, 8 lru
search space	6.236.703	18

constraints
IF working set is small THEN size ≤ 32
IF locality is big THEN line size ≥ 16
line size ≤ size
IF locality is big THEN associativity ≤8
power_of_2(size) power_of_2(size) power_of_2(associativity)
IF line size >= 16 THEN associativity >1
IF associativity = 1 THEN replacement = no ELSE IF associativity ≤ 8 THEN replacement = LRU

Tab. 1: concrete example of search space restriction, parameter domains before and after restriction by constraints

4.3 Alternative Optimization Strategies

The system SPEISE as a whole configures and optimizes a memory system according to several objectives. As has been pointed out in the previous subsection this task is performed by a heuristic search strategy that utilizes domain specific search control knowledge for parameter labeling and redesign.

There are several well-known alternative optimization strategies. *Evolution strategies* [Schw81] consider several candidates in parallel. New candidates are created by syntactic, domain-independent mutations of existing ones. An evaluation function selects the most promising candidates to be considered further on. This method works well, if a large set of candidates can be created and evaluated with limited computational effort. As the evaluation of memory configuration involves time-consuming simulation runs, this condition does not hold for memory synthesis.

Integer linear programming [Neum75] also provides an optimization strategy. In contrary to CLP it is limited to numeric parameter types. The method requires a target function that can be used to determine an (optimal) solution in a single step. In memory synthesis, an optimal solution cannot be determined in a single step, as an appropriate compound target function is lacking. Instead, simulation runs are required to evaluate the quality of a candidate to be modified later. Therefore integer linear programming is not suitable for memory synthesis.

CYCLOPS [Navi91] also uses constraints to represent conditions and dependencies in the considered domain. The system uses a modified A* search algorithm to determine a set of pareto optimal solutions. Like the other approaches mentioned above, CYCLOPS does not provide a redesign mechanism that modifies a candidate in a specific way, according to the results of an analysis. This feature is indispensable for memory synthesis.

5. The SPEISE System

SPEISE designs a memory system for a given general purpose computer architecture and a class of programs representing the typical applications on this computer architecture. The memory system is optimized according to several given objectives. Fig. 3 shows the components of SPEISE:

In the following, we will first describ the cooperation of SPEISE's components. Afterwards particular aspects will be pointed out in detail.

5.1 The Design Cycle

The input data for SPEISE are memory reference sequences of application programs, architectural features and objective weights denoting their im-portance for optimization.

Before starting the design cycle, memory reference sequences of application programs are analyzed to examine their typical features ('trace analysis').

Constraints sets expressing the relations between input data and design decisions are selected as a result of an analysis of the input data (i.e., constraints set for single/multi-processor systems).

At the beginning of each design cycle, the selected constraints are imposed to restrict the search space ('restriction by constraints').

After this step, decision making starts ('parameter labelling'). The order of making decisions is determined by a domain specific rating of each decision. The rating reflects the impact of these design decisions on the performance. For instance, the most important design decisions for a cache are size, block size and associativity. The decision process takes

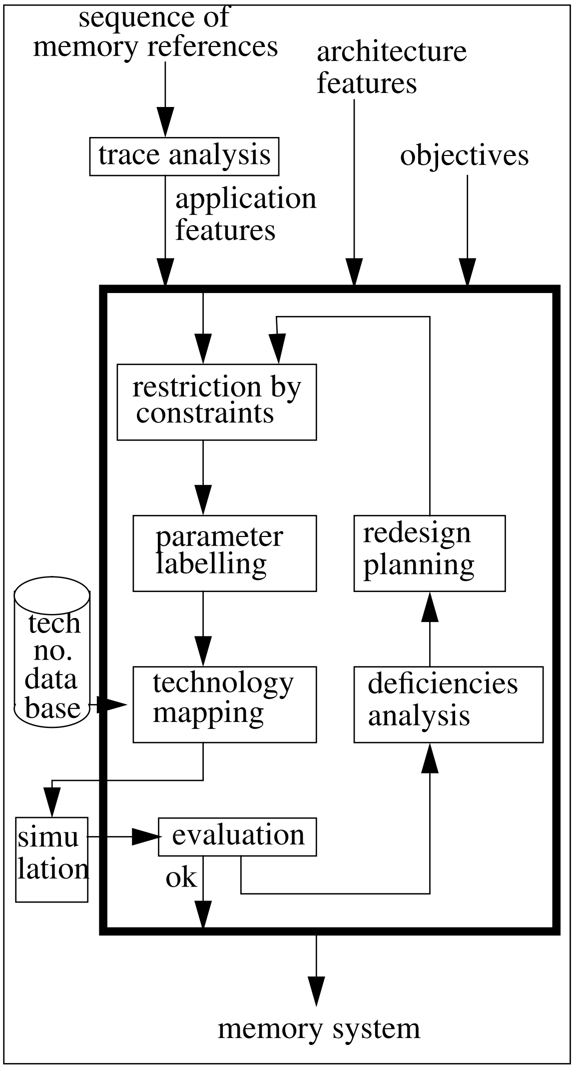

Fig. 3: components of the SPEISE

advantage of the restrictions to the search space caused by the design constraints. Decisions are made with respect to the objectives. For instance, if the objective cost is to be minimized, the size of a cache is set to the minimum in the remaining domain.

Components of the designed memory system are mapped to existing on-chip and off-chip memory modules described in the technology database (‘technology mapping’).

An event-driven trace simulation is used to imitate the accesses of the application programs to the memory system (‘simulation’). During this simulation values for performance criteria like average access time and different miss rates for caches are calculated.

Using the simulation results, the ‘evaluation’ component evaluates the synthesized memory configuration. If the quality of the designed memory system meets a calculated threshold value for each of the objectives, the memory system is accepted and presented to the user. Otherwise redesign has to be performed. This is the typical case for the first design cycle.

If the design is not accepted, the ‘deficiencies analysis’ examines which of the threshold values are missed. For each of the missed values possible revisions of design decisions are determined(i.e., if the miss rate of a cache is to high, an increase in size or associativity is proposed).

‘Redesign planning’ examines the list of possible changes proposed by the deficiencies analysis. As these revisions may be contradictory, a consistent subset of them is selected. This process is guided by heuristic rules. The changes are expressed in form of constraints.

To avoid recreations of previous designs, the ‘deficiencies analysis’ and the ‘redesign planning’ take redesign actions of previous design cycles and their consequences on the performance into account.

At the beginning of the next design cycle the constraints created by the ‘redesign planning’ are imposed. The cycle is continued with imposing the constraints in the ‘restriction by constraints’ step.

Redesign stops if the ‘evaluation’ component or the designer (user of the tool) accept the designed memory system.

Fig. 4 shows the algorithm for SPEISE’s heuristic search strategy from an imperative point of view. The cooperation of the implicit control strategy available by use of declarative programming and the explicit control actions performed by the components of SPEISE is described. The outer WHILE loop guides the redesign cycle by use of a new non chronological backtracking strategy: restrict the search space by imposing constraints, label the parameters, simulate and evaluate the resulting memory configuration, and relax constraints or plan redesign if necessary. In the inner WHILE loop the memory parameters are labeled one after the other (‘parameter labelling’). This step is completely guided by the chronological backtracking mechanism of declarative programming. The relaxation of constraints is guided by a separate control strategy.

5.2 Selected Aspects

The following selected aspects are pointed out in detail:

- *Provision of Memory Reference Sequences*
  It is difficult to get adequate memory reference sequences. If a memory system for an existing computer architecture is improved, an existing compiler can be used to generate memory refer-

```
search strategy(Application Features, Architectural Features, Objectives)
BEGIN
    Constraints_perm := <set of maximal constraints, permanently available>;
    Constraints_redesign := <empty>;
    <initialize Mem_Config with set of unlabeled memory parameters>;
    WHILE (<Mem_Config not acceptable>)
    DO
        <impose Constraints_redesign>;
        <impose Constraints_perm>;
        WHILE (<there are unlabeled parameters in Mem_Config>
                AND <restricted search space not completely searched>)
        DO
            <select next parameter P_i to be labeled>;
            <label P_i and resume corresponding constraints>;
            IF (<there is no value for P_i consistent to Constraints_perm and Constraints_redesign>)
            THEN
                <chronological backtracking to labelling of the previous parameter P_i-1>;
                <select a different value for P_i-1 >;
            FI
        OD
        IF (<no consistent labelling for all parameters in Mem_Config found>)
        THEN
            <relax Constraints_perm>;
        ELSE
            <simulate and evaluate Mem_Config>;
            IF (<Mem_Config not acceptable>)
            THEN
                <plan redesign operations>;
                Constraints_redesign := <domain restrictions according to redesign operations>;
            FI
        FI
        Mem_Config := <set of unlabeled memory parameters>;
    OD
END
```

Fig. 4: algorithm of SPEISE's heuristic search strategy

ence sequences from application programs.

If the given computer architecture is new but similar to another one (i.e., to a predecessor model) in terms of address generation, memory reference sequences of the predecessor can be used.

An address sequence can also be made independent of poor compilation [McNi88].

Another possibility is the generation of memory reference sequences. This can be done, if the features (locality of reference, size of working set, etc.) of the application programs are known [Hyat93; McNi88]. In [Hoba89] some features of symbolic programs are described. SPEISE offers a *generator* that creates memory reference sequences according to given features.

Notice, that the features, relevant for memory synthesis, like locality of reference and working set, depend rather on the application program than on the computer architecture and compiler. Therefore it is feasible to take reference sequences of similar systems or to generate them.

- *Inconsistencies in the Set of Constraints Restricting the Search Spac*e
  The set of constraints in the restriction component has been derived from an intensive analysis of the relevant literature [Arar89, Henn96, Przy90, Smit92, etc.]. Each constraint is marked by a weight denoting its importance. As described above the set of constraints may be inconsistent because it is derived from different analyses and

quantified heuristically. If imposing of several constraints results in an inconsistency, the less important ones are relaxed, to get a consistently restricted search space.

- *Technology Adaptation*
  As mentioned above, the memory system under design is mapped to existing memory modules. For this purpose a technology database is used, that contains technology dependent information about memory modules. For on-chip modules formulas calculating the area and access times are given, depending on the component size. For off-chip modules size, access time, and cost are given. It is easy to adapt SPEISE to new IC technologies because only the information in the technology database has to be changed or extended.
- *Combination of Objectives*
  In SPEISE it is possible to optimize the memory system according to more than one objective. The importance of each objective can be expressed by a weight. SPEISE optimizes the memory system according to these user supplied weights. Based on these weights a threshold for each objective is calculated. If all thresholds are met, thresholds for the most important objectives are tightened successively as much as possible.
- *Alternatives in Redesign Planning*
  SPEISE can select between two modes of redesign planning. In stepwise mode redesign planning selects exactly one parameter to change. In multistep mode several changes of parameters are performed in one redesign cycle.

5.3 Organization of the System as an Intelligent Synthesis Assistant

The system SPEISE does not aim at a complete automation of the memory synthesis process. This would not be adequate for this complex high-level synthesis task without standardized synthesis methodology. Instead, SPEISE is organized as an intelligent synthesis assistant that supports an interactive design style. By use of the assistant, the designer can create, evaluate and compare several design alternatives quickly.

The user can make decisions and limit the search space in that way. Decisions performed by the system can be changed easily. The consequences of these changes on other aspects of the memory architecture under design are propagated automatically by the assistant. In this interactive mode, SPEISE can be used as an intelligent “editor” that enables a flexible selection resp. modification of design decisions and responds by showing the side effects of these actions. Based on these results the user can accept the intermediate memory configuration or perform further design changes.

5.4 Implementation

The main part of SPEISE (framed by a bold painted rectangle in Fig. 3) is implemented in ECLiPSe, a CLP language from the ECRC [ECLI95], on a SUN workstation. The first prototype is restricted to single processor systems and focuses on synthesizing the cache and TLB hierarchy. As mentioned above, this is the main task of memory synthesis for general purpose processors. Main memory design decisions, given by the designer, are taken into account. The handling of compound objectives is simplified in the implemented prototype. The memory system is optimized primarily according to the most important objective. Nevertheless the other objectives have an impact on some design decisions leading to the final memory configuration.

The components ‘trace analysis’ (including the memory reference sequence generator) and ‘simulation’ are implemented in C++.

The prototype of SPEISE has been implemented by graduated students in an 1-year project [SPEI95].

6. Results

SPEISE has been used to design cache and TLB hierarchies for several computer architectures. In most cases few redesign cycles (5 to 10) were sufficient to (re)design a configuration that meets the performance thresholds.

Due to the lack of space, only the key features of the design process and the results are described in following example.

6.1 Example

The example shows the design of a cache hierarchy for a computer architecture similar to a SUN SPARCstation (Tab. 2) with a main memory given in Tab. 3. Tab. 4 shows the features of a class of application programs running on this computer architecture.
They have been extracted by the ‘trace analyzer’ separately for instruction, data, and mixed references.

Tab. 5 shows the cache configuration designed by SPEISE after the first synthesis cycle. An accepted

clock frequency (MHz)	33
pipeline stages	4
data bus size (bit)	64
address bus size (bit)	32
...	

Tab.2: architectural features

size (MByte)	32
page size (KByte)	8
organization	segmentation
interleaving degree	8
...	

Tab. 3: main memory features

	mixed	instr.	data
spacial locality	big	big	small
working set	small	small	medium
frequency of ref.	small	small	small
number of reads	big	big	medium
variance	small	small	small
...			

Tab. 4: application program features

end configuration was reached after seven design cycles. The differences between the final and the first configuration are shown by the values in parentheses.

	1. I-cache	1. D-cache	2. cache	...
adressing	virtual	virtual	real	
size (KB)	16	32	512	
block size (b)	16	64	128	
associativity	**8 (full)**	**2(16)**	2	
replace. strat.	lru	lru	lru	
prefetching	tagged	tagged	tagged	
write strat.	-	copy b.	copy b.	
...				

Tab.5: memory configuration after the 1. cycle and the end configuration in parenthesis

For each redesign cycle SPEISE performs a simulation of a trace according to the application parameters, an evaluation of the simulation results, a deficiencies analysis to find a set of cache parameters as candidates for modification, and redesign planning to select parameters to change and to determine the extent of the changes.

Then the next design cycle is started to accommodate the other cache parameters. Tab. 6 gives the changes for each redesign cycle.

	changes in the redesign plan (and implied changes)
1.	increase the associativity of the instruction cache from 8 to 16
2.	increase the associativity of the data cache from 2 to 4
3.	increase the associativity of the data cache from 4 to 8
4.	decrease the block size of the instruction cache from 16 to 8
5.	increase associativity of instruction cache from 16 to full (block size of the instruction cache increases from 8 to 16)
6.	increase the associativity of the data cache from 8 to 16

Tab. 6: Changes for each redesign cycle

For instance, during the first redesign cycle (design cycle 2) the associativity of the instruction cache has been changed from 8 to 16.

Tab. 7 shows some of the performance parameters measured by the simulator after each design cycle: average access time and average values for hit time and miss ratio of the first level caches.

design cycle no	av. access time (nsec)	hit time (nsec)	miss ratio (%)
1	30.53	19.40	30.46
2	30.52	19.40	30.46
3	35.39	20.00	18.07
4	34.60	20.75	22.56
5	34.61	20.76	22.56
6	34.72	22.26	22.56
7	35.93	23.01	4.60

Tab 7: performance parameters after each design cycle

The performance thresholds to be met are calculated by SPEISE according to the input parameters (see above) and the objectives (here access time). For instance, the hit time threshold has to be below the clock time (here a clock frequency of 33 Hz implies a clock time of 30 nsec).

After the first design cycle both thresholds for hit time and average access time are met, but the miss ratio, which should be less than 5%, is much to high. To reduce the miss ratio, the 'deficiencies analysis' proposes to increase the associativity or the size of one of the caches. The 'redesign planner' decides to increase the associativity of the instruction cache to a value greater than 8. In the next 'restriction by constraints' and 'parameter labelling' step the value is set to 16. This is done due to some further analysis of the simulation results (detailed miss ratios for each cache like capacity or conflict miss ratio, etc.). The improvements are minimal. Therefore in the next redesign cycle the associativity of the data cache is increased.

After the seventh design cycle the miss ratio finally meets the corresponding threshold. The average access time and hit time have been increased but still meet their thresholds. So the cache configuration is presented to the designer and if he/she accepts the design the system stops. Otherwise he/she can propose changes to one or several design parameters and the system starts the next redesign cycle.

In this simple example it is adequate to select changes only for associativity and block size of the first level data and instruction cache because the miss ratio is decreased successively without increasing the time parameters to an unacceptable value. In other examples SPEISE selects a greater variety of redesign operations.

6.2 Multi-step mode

The described example was designed in the stepwise mode: For each redesign cycle exactly one parameter has been changed (and the others were readjusted). In this way the effect of this redesign step can be observed and it can be decided if this step leads into the right direction. This is important for further redesign cycles. In multistep mode SPEISE often terminates successfully in fewer redesign cycles than in stepwise mode. In multistep mode SPEISE synthesizes a memory configuration for the described example that meets nearly all thresholds after the second design cycle. But it is difficult to decide which of the changes were successful because their effects can nullify each other. This is in particular the case in the domain of memory synthesis, because the dependencies between the design parameters and performance parameters as a whole are not quantified (see above) and the synthesis process is guided by heuristics and evaluated by simulation. In the stepwise mode one parameter after the other can be changed and the known qualitative dependencies between parameters (as described in literature) can be used to do this. As a consequence, both modes can be combined: For the first three redesign cycles multistep is used. If it does not lead to success, the system switches to stepwise mode.

6.3 Future Work

Currently, the implemented prototype is enhanced and extended in different ways:

- One aspect is the *improvement of the constraints*. Based on the evaluation of the performed system runs, additional constraints are formulated.
- A *hierarchical organization* of the constraint set will enable an improved constraint relaxation strategy.
- To enable an adequate treatment of trade-offs, currently a more elaborated *handling of multiple objectives* is developed.
- The most important current work is related to the range of applicability for the system. SPEISE is extended to *memory architectures for multiprocessor systems* including main memory synthesis.

7. Conclusion

We have presented a first approach to CAD support for memory synthesis for general purpose processors. The implemented prototype aims at "closing a gap" in the current tool support for general purpose computers. It provides a systematic synthesis methodology and demonstrates that tool support can be achieved for this complex task. The system SPEISE does not aim at complete automation of memory synthesis. Instead, it is organized as an intelligent synthesis assistant system that supports an interactive design style. By use of the assistant, the designer can create, evaluate and compare several design alternatives quickly.

The synthesis of memory systems for general purpose computers is a multidimensional optimization problem. Objectives like memory performance and cost have to be considered. Besides that, the task is characterized by numerous influencing factors and design decisions to be made. Memory synthesis can be formalized as a *"parameter selection problem"* [Navi91]. Constraint logic programming is an adequate programming paradigm for this application. Logic programming enables the quick development of a compact prototype and does not require the detailed formulation of program control, a characteristic requirement of most other programming styles. By use of constraints an adequate, explicit representation of the exact resp. heuristic dependencies between influencing factors and design decisions can be formulated. Constraint weights express the confidence in heuristic dependencies and guide the relaxation of an overconstrained design state.

Memory synthesis for general purpose computers must depend on the context and the purpose of the computer architecture. The organization of the system SPEISE reflects this crucial requirement. Characteristic features of the computer architecture can be provided by the user as input information for the synthesis task. Besides that, information about the class of application programs for the computer to be designed is considered. In this way, the design can be tailored to a narrow resp. broad class of application programs.

Acknowledgments

We are grateful to P. Marwedel and U. Bieker for their corrections and improvements of this paper. Further thanks goes to I. Bebic, H. Drumann, O. Hesse, R. Jäger, B. Jeussmann, I. Kaleja, I. Krämer, M. Oelbracht, Th. Schulte, F. Wiechers, and M. Wojciechowski for fecund discussions and for implementing the SPEISE system.

8. Literature

[Agar89] A. Agarwal, M. Horowitz, J. Hennessy. An Analytical Cache Model. ACM Transactions on Computer Systems, May 1989.

[Aho86] A.V. Aho, R. Sethi, J.D. Ullmann. Compilers: Principles, Techniques and Tools. Reading, Ma: Addison-Wesley, 1986.

[Bala88] M. Balakrishnan, A.K. Majumdar, D.K. Banerji, et al. Multiple Storage Adaptive Multi-Trees. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, March 1988.

[Berg93] J. van den Berg, D. Towsley. Properties of the Miss Ratio for a 2-Level Storage Model with LRU or FIFO Replacement Strategy and Independent References. IEEE Transactions on Computers, April 1993.

[ECLI95] ECLIPSE 3.5. ECRC Common Logic Programming System - User Manual -. ECRC GmbH Munich Germany, December 1995.

[Frue93] Th. Frühwirth, A. Herold, V. Küchenhoff, et al. Constraint Logic Programming - An Informal Introduction. Technical Report ECRC-93-5. ECRC Munich Germany, February 1993.

[Gajs92] D.D. Gajski, N.D. Dutt, A. Wu, St. Lin. High-Level Synthesis: Introduction to Chip and System Design, Kluwer Academic Publishers, 1992.

[Henn96] J. L. Hennessy, D. A. Patterson. Computer Architecture: A Quantitative Approach, 2nd Ed. Morgan Kaufman, 1996.

[Hoba89] W.C. Hobart Jr., H.G. Cragon. Locality Characteristics of Symbolic Programs. In IEEE International Conference on Computer Design, pages 508 – 511, 1989.

[Hyat93] C. Hyatt. A High Performance Object-Oriented Memory. In Computer Architecture News, 1993.

[Kolk93] T. Kolks, B. Lin, H. De Man. Sizing and Verification of Communication Buffers for Communicating Processes. In IEEE/ACM International Conference on Computer-Aided Design, 1993.

[Kris96] C. M. Krishna. Performance Modeling for Computer Architecture. IEEE Computer Society Press, 1996

[McFa90] M. C. McFarland, A. C. Parker, R. Camposano. The High-Level Synthesis of Digital Systems. In Proceedings of the IEEE, February 1990.

[McNi88] G.D. McNiven, E.S. Davidson. Analysis of Memory Referencing Behavior for Design of Local Memories. Computer Architecture News, May 1988.

[Navi91] D. Navinchandra. Exploration and Innovation in Design - Towards a Computational Model, Springer 1991.

[Neum75] K. Neumann. Operations Research Verfahren I - III (German), Carl Hanser, 1975.

[Przy90] St. A. Przybylski. Cache and Memory Hierarchy Design: A Performance Directed Approach. Morgan Kaufmann 1990.

[Rau91] B.R. Rau. Pseudo-Randomly Interleaved Memory. Annual International Symposium on Computer Architecture, pages 74–83, 1991.

[Saav95] Measuring Cache and TLB Performance and Their Effect on Benchmark Runtimes. IEEE Transactions on Computers, October 1995.

[Schw81] H.-P. Schwefel. Numerical Optimization of Computer Models. Chichester: Wiley, 1981.

[SPEI95] R. Beckmann, J. Herrmann (ed.).Endbericht der Projektgruppe SPEISE. Internal Report. University of Dortmund, 1995. (in German)

[Smit82] A.J. Smith. Cache Memories. ACM Computing Surveys, September 1982.

[Wilk95] M. V. Wilkes. The Memory Wall and the CMOS End-Point. Computer Architecture News, September 1995.

[Wulf95] W.A. Wulf, S. A. Mokee. Hitting the Memory Wall: Implications of the Obvious. Computer Architecture News, March 1995.

[vMee92] J. van Meerbergen, P. Lippens, et al. Ar-chitectural Strategies for High-Throughput Applications. Journal of VLSI Signal Processing, 1992.