STOREQ: **STO**rage **REQ**uirement Estimation and Optimization tool for Data Intensive Applications

Per Gunnar Kjeldsberg, Einar J. Aas
Norwegian University of Science and Technology
Trondheim, Norway
pgk/aas@fysel.ntnu.no

Francky Catthoor
IMEC, Leuven, Belgium
Also at EE.Dept. of Kath. Univ. Leuven
catthoor@imec.be

Martin Palkovic
IMEC, Leuven, Belgium
palkovic@imec.be

We demonstrate the STOREQ tool for STOrage REQuirement estimation and optimization of data intensive applications. STOREQ can guide the designer during the early system design steps towards an implementation with low memory usage.

Many integrated circuit systems, especially in the multimedia and telecom domains, are inherently data dominant. For this class of applications, data transfer and storage largely determine cost and performance parameters. This is the case for *chip size*, since large memories are usually needed, *performance*, since accessing the memories may very well be the main bottleneck, and *power consumption*, since the memories and buses consume large quantities of energy. Even for systems with caches, the overall storage requirement has vital impact on the performance and power consumption, since it greatly influences the number of slow and power expensive cache misses. During the system development process, the designer must hence concentrate first on exploring the data transfer and storage to produce a cost-optimized end product. The STOREQ tool assists the designer during these early steps of the system design.

At the system level, no detailed information is available regarding the size of the memories required for storing data in the alternative realizations of an application. To guide the designer and assist in selecting the best solution, estimation techniques for the storage requirements are needed, very early in the system design trajectory. The simplest estimates use the maximal size of the intermediate array data as declared in the application code. This is however not representative for the effective size required for their storage during the actual execution since arrays and parts of arrays may not be alive simultaneously. To achieve accurate estimates, the so-called in-place mapping opportunity generated by these non-overlapping lifetimes must be taken into account. For scalars, a relatively simple lifetime analysis suffices, but for arrays, this is extremely complex due to the huge number of signals and the often very complex interdependencies between them.

For our target classes of data dominant applications the high-level description is typically characterized by large multi-dimensional nested loops and arrays. Within the loop nests statements access the arrays using read and write operations. At the beginning of the design process, no information about the execution order of these loops is available, except what is given from the data dependencies between the statements in the code. As the process progresses, the designer makes decisions that gradually fix the ordering, until the full execution ordering is known. This execution ordering determines the lifetimes of the array elements, and hence the storage requirements of the arrays. To guide the designer it is therefore essential to have storage requirement estimates that can take the available partially fixed execution ordering into account during exploration. Previous work has either not taken execution ordering into account at all, resulting in large overestimates, or required a fully specified ordering. In the last case, a full exploration of all alternative orderings of the unfixed loop dimensions is needed, which is infeasible for fast feedback purposes.

The storage requirement estimation methodology implemented in STOREQ solves these important design problems. The methodology is divided into four steps. In the first step, a data-flow graph is generated that reflects the data dependencies in the application code. The array accesses and the dependencies between them are described using a polyhedral model. The second step places the polyhedral descriptions of the array accesses and their dependencies in a so-called common iteration space. The third step estimates the upper and lower bounds on the storage requirement of individual data dependencies in the code, taking into account the available execution ordering. As the execution ordering is gradually fixed, the upper and lower bounds on the data dependencies converge. Finally, simultaneously alive data dependencies are detected. Their combined maximal size at any time during execution equals the total storage requirement of the application. An important part of the estimation technique utilizes loop ordering guidance to estimate upper and lower bounds on dependency sizes.

The STOREQ tool has been developed using MATLAB, and must be run on the MATLAB platform. It takes as input a polyhedral description of the array accessing and dependencies in the application. A prototype interface exists between STOREQ and the ATOMIUM tool being developed at IMEC. This enables automatic generation of the polyhedral input description from the original application code. The focus of the current version of the tool is on the third step of the methodology. The main output is hence upper and lower bounds on the storage requirement of individual dependencies, given a partially fixed execution ordering specified by the designer. In addition, STOREQ employs the guiding principles of the methodology on a more global basis and suggests an ordering for the still unfixed dimension. The demonstration at DATE University Booth shows how the designer can use these parts of STOREQ to achieve an implementation with low memory requirements. Both small artificial and large real life examples are used.

Further reading:
- http://www.fysel.ntnu.no/~pgk/STOREQ-Users-Manual.pdf
- http://www.imec.be/atomium/
- Kjeldsberg et at.: Automated data dependency size estimation with a partially fixed execution ordering, ICCAD-2000
- Kjeldsberg et at.: Detection of partially simultaneously alive signals in storage requirement estimation for data-intensive applications, DAC-2001