STOREQ: STOrage REQuirement Estimation and Optimization tool for Data Intensive Applications

Per Gunnar Kjeldsberg, Einar J. Aas
Norwegian University of Science and Technology, Trondheim, NORWAY

Francky Catthoor
IMEC, Leuven, BELGIUM
Also at Katholieke Universiteit Leuven

Martin Palkovic
IMEC, Leuven, BELGIUM

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.

Motivation and Context

For data dominated electronic systems:

- Data transfer and storage determine cost and performance parameters.
- It should be the main focus of the designer to achieve cost-optimized end product [Catthoor98].

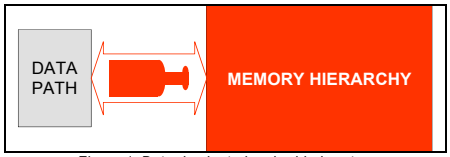

Figure 1: Data dominated embedded system

- Code description is characterized by large multi-dimensional loop nests and arrays.

```
for (y_s=0; y_s<=31; y_s++) {
  for (x_s=0; x_s<=31; x_s++) {
    for (y_p=0; y_p<=15; y_p++) {
      for (x_p=0; x_p<=15; x_p++) {
        if ((x_p == 0)&(y_p == 0)) sad[y_s][x_s][y_p][x_p]=
                                   f(curr[y_p][x_p], prev[y_s+y_p][x_s+x_p]);
        else if ((x_p == 0)&(y_p != 0)) sad[y_s][x_s][y_p][x_p]=
                                   g(sad[y_s][x_s][y_p-1][15], curr[y_p][x_p],
                                                        prev[y_s+y_p][x_s+x_p]);
        else sad[y_s][x_s][y_p][x_p]= g(sad[y_s][x_s][y_p][x_p-1],
                                   curr[y_p][x_p], prev[y_s+y_p][x_s+x_p]);
} } } }
```

Figure 2: Code example (MPEG-4 motion estimation kernel)

- At the system level there is no detailed storage requirement information available
- Estimates are hence essential and must take in-place mapping opportunities into account.
- Storage requirement is mainly decided by the ordering of the loops surrounding the arrays.
- Designers gradually fix execution ordering guided by novel optimization principles.
- STOREQ provides converging estimates of upper and lower bounds on the storage requirement, given the partially fixed execution ordering.
- Previous work either assumes a fully fixed ordering, e.g. [Zhao99], or does not take it into account at all [Balasa95].

Estimation Methodology

Generate data-flow graph reflecting data dependencies in the application code

↓

Position DPs in common iteration space according to dependencies and partially fixed execution ordering

↓

Estimate upper and lower bounds on dependency sizes of individual dependencies based on partially fixed execution ordering

↓

Find simultaneously alive dependencies and their maximal combined size

=

Bounds on the application's storage requirement

Concept Definitions

Iteration Space:
The iteration nodes over which the iterators of a loop nest run. See Figure 3.

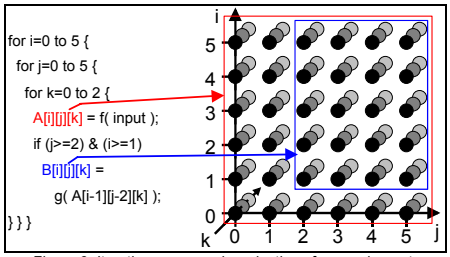

Figure 3: Iteration space and production of array elements

Dependency Part (DP):
The iteration nodes where array elements are written by one statement that are later read by a depending statement.

Dependency Vector (DV):
Connects write and read nodes.

Dependency Vector Polytope (DVP):
Polytope spanned by the DV.

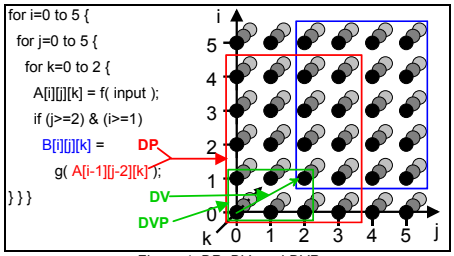

Figure 4: DP, DV, and DVP

Spanning/Nonspanning Dimensions (SD/ND):
The iteration space dimensions that are/are not a part of the DVP. In Figure 4: SD={*i,j*}, ND={*k*}.

Data-flow Graph

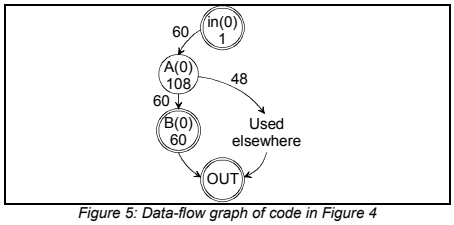

Figure 5: Data-flow graph of code in Figure 4

Individual Dependency Sizes

Estimate with no execution ordering fixed:
Upper Bound (UB) = Size (DP) - Overlap = 36
Lower Bound (LB) = Size (DVP) - Overlap = 5

	Fixed innermost	Fixed outermost
SD	*Small* increase in LB *Big* reduction in UB	*Big* increase in LB *Small* reduction in UB
ND	*Big* increase in LB Unchanged UB	Unchanged LB *Big* reduction in UB

Figure 6: Typical consequences of fixation

Estimate with ND *k* outermost:
DP reduced → UB=12. DVP unchanged → LB=5. See Figure 7.

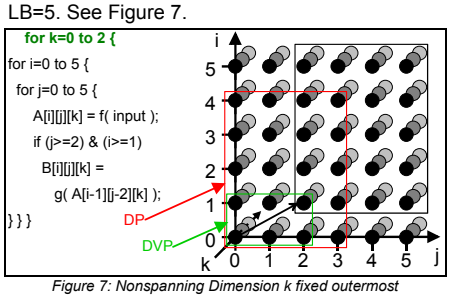

Figure 7: Nonspanning Dimension k fixed outermost

Estimate with SD *i* second outermost:
DP reduced → UB=6. DVP extended → LB=6. See Figure 8.

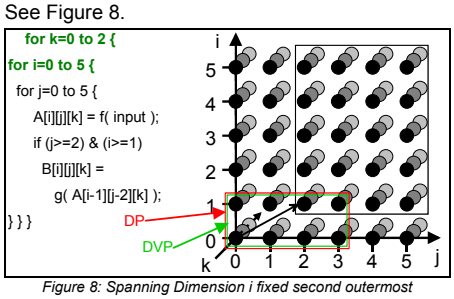

Figure 8: Spanning Dimension i fixed second outermost

Guiding Principles

- Fix SDs innermost in loop nest
- Order SDs according to *Length Ratio* (LR)
- $LR_j > LR_i \rightarrow j$ inside i

$$LR_{d_i} = \frac{|DVP_{d_i}| - 1}{|DP_{d_i}| - 1}$$

$LR_i = 1/4$
$LR_j = 2/3$

Optimal ordering Figure 4: *k i j* (UB = LB = 6)

The STOREQ Tool

- Interfaced with mature Atomium tool for data-flow graph and iteration space generation
- Current version focused on size estimation for individual dependencies
- Uses guiding principles with more global perspective and suggests ordering for unfixed dimensions

Experimental Results

MPEG4 Video Motion Estimation

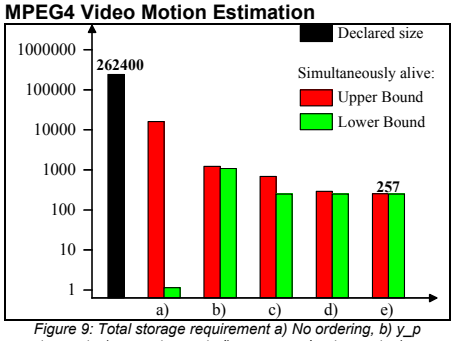

Figure 9: Total storage requirement a) No ordering, b) y_p outermost, c) y_s outermost, d) x_s second outermost, e) y_p third outermost and x_p fourth outermost

Cavity Detection

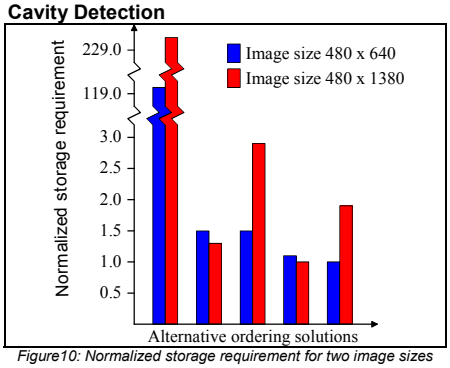

Figure10: Normalized storage requirement for two image sizes

Presented at Design Automation and Test in Europe, DATE 2002, Paris, March 2002

imec NTNU