DRAM or no-DRAM? Exploring Linear Solver Architectures for Image Domain Warping in 28 nm CMOS

Michael Schaffner*†, Frank K. Gürkaynak*, Aljoscha Smolic†, Luca Benini*‡
*ETH Zürich, 8092 Zürich, Switzerland, †Disney Research Zurich, Switzerland, ‡Università di Bologna, Italy

Abstract— **Solving large optimization problems within the energy and cost budget of mobile SoCs in real-time is a challenging task and motivates the development of specialized hardware accelerators. We present an evaluation of different linear solvers suitable for least-squares problems emanating from image processing applications such as image domain warping. In particular, we estimate implementation costs in 28 nm CMOS technology, with focus on trading on-chip memory vs. off-chip (DRAM) bandwidth. Our assessment shows large differences in circuit area, throughput and energy consumption and aims at providing a recommendation for selecting a suitable architecture. Our results emphasize that DRAM-free accelerators are an attractive choice in terms of power consumption and overall system complexity, even though they require more logic silicon area when compared to accelerators that make use of external DRAM.**

I. Introduction

With the advent of powerful *Systems on Chip* (SoCs) and specialized co-processors, it has recently become feasible to execute a variety of video stream analysis and synthesis algorithms on mobile devices. An example for a state-of-the-art hardware platform is the Google *Tango*, which is a mobile device able to run high quality *Simultaneous Localization and Mapping* (SLAM) algorithms, such as [1], in real-time. A key enabler for such technologies are mobile GPUs and specialized co-processors, such as the *Myriad-1* and *Myriad-2* from Movidius.

Apart from SLAM, other popular application examples are computational photography- and computer vision problems such as *high dynamic range* (HDR) compression, optical flow estimation and inpainting [2][3], or *Image Domain Warping* (IDW) applications such as *video retargeting*, *stereo remapping* and *stereo to multiview conversion* [4–6]. What all these applications have in common is that they perform an optimization step where certain quantities (such as camera-/object positions and pixel values/-coordinates) are optimally estimated given multiple measurements, e.g. the images or extracted image features. Such optimization problems are often posed as least-squares (LS) problems, which essentially boil down to solving large, sparse systems of linear equations. Solving these problems in real-time within the power budget of mobile SoCs is challenging. This motivates the development of custom hardware accelerators which are able to solve common subtasks such as LS with very high energy efficiency.

In this paper, we present an architectural exploration of iterative and direct linear solvers for LS problems emanating from IDW applications such as [4–6]. In particular, the aim is to compare the energy efficiency of solver variants that make use of off-chip (DRAM) memory, and solvers that use on-chip SRAM only. This is a very important design aspect, since the use of a DRAM based solution has system-level implications in terms of modularity (the overall bandwidth has to be shared among all subsystems in an SoC), energy efficiency and additional cost such as memory controller IP, DRAM components and I/Os. In this light, it may be more attractive to use a solution which is suboptimal in terms of required on-chip area, but which does not use any DRAM. A basic estimation framework

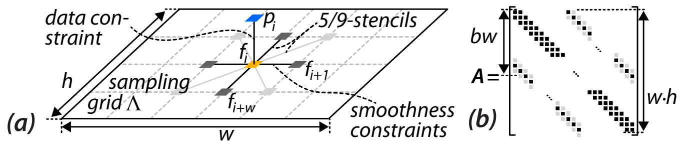

Fig. 1. a) Sampling grid with unknowns f_i and constraints on a 5-stencil. b) Sparsity structure of the resulting matrix A.

has been developed to compare these architecture variants in terms of energy consumption, circuit area, throughput and precision. We highlight potential tradeoffs to guide architectural and system level decisions, and show that in 28 nm CMOS technology, complete on-chip solvers are perfectly feasible for matrix dimensions up to 128 k (which enables high quality synthesis of full-HD video in real-time).

The remainder of the paper is organized as follows: Sections II and III summarize related work and some preliminaries. The evaluated hardware architectures are explained in Section IV, our evaluation framework is summarized Section V, and experimental results are given in Section VI. Finally, Section VII concludes the paper.

II. Related Work

The most common iterative and direct solvers used in LS problems are *conjugate gradient* (CG) and *Cholesky decomposition* (CD) based solvers, since they exploit the mathematical properties of such problems in order to achieve high energy efficiency [7][8]. The design of hardware architectures for these iterative and direct solver types has been addressed in [9][10] and [11–15], respectively. In all of the listed work, FPGA targets are considered, and most use relatively small, non-sparse matrices - except Greisen et.al. [10] and Schaffner et. al. [15]. In the former, direct CD and iterative CG solver architectures for large, sparse, linear systems in image and video processing applications are developed and implemented on an FPGA. In the latter it is shown that in the case of IDW, an incomplete CD can provide an approximate solution that is still accurate enough - yet with significantly lower computational cost. These two serve as a basis for the hardware architectures evaluated in this work.

As opposed to previous work, this paper focuses on ASIC implementations of such architectures and provides an exploration in 28 nm CMOS technology. Our estimation framework is based on analytical architecture and activity models, and a set of pre-characterized submodules (such as floating point operators). An example for a similar framework is the one by Vishnoi et. al. [16] which is used to analyze different configurations of QR-decomposition accelerators. A main difference with respect to their work is that our designs are significantly larger and also comprise SRAM macros and DRAM interfaces which are a key part of the design and must be accounted for. In this work, the estimation for the external DRAM components are based on the *Micron System Power Calculator*[1]. Other popular estimation frameworks are CACTI [17] and McPAT [18], but they are not suitable for our application since they focus on general purpose cache- and multi-processor architectures.

[1]`www.micron.com/products/support/power-calc`

III. Preliminaries

A. Sparse Linear Systems in Image and Video Processing

Many image and video processing algorithms can be posed as a quadratic minimization problem of the form

$$\min_{\mathbf{f}} \left(E\left(\mathbf{f}\right)\right) = \min_{\mathbf{f}} \left(E_d\left(\mathbf{f}\right) + E_s\left(\mathbf{f}\right)\right), \tag{1}$$

where the data term E_d enforces function values at certain sampling positions, and the smoothness term E_s is a regularizer that propagates the known sample values to adjacent sampling positions. The vector $\mathbf{f}$ is holding the samples of an unknown discrete function (e.g. pixel intensities or coordinate values) defined on a two-dimensional sampling grid Λ with width w and height h. The two terms E_d and E_s are briefly summarized below and a more detailed description can be found in [6][10][19].

Let i be the *linear* index of the sampling points of the grid Λ with $i \in \mathcal{D}_\Lambda = \{1, 2, ..., w, ..., w{\cdot}h\}$. Figure 1 a) illustrates the introduced relations for one sample f_i of $\mathbf{f}$. The data term usually has the form

$$E_d\left(\mathbf{f}\right) = \sum_{i \in \mathcal{D}_\Lambda} \lambda_i \left(f_i - p_i\right)^2,$$

where p_i are the constraint values, and the parameters λ_i are weights indicating the relative importance of the corresponding constraints. The smoothness term contains differential constraints defined among neighboring samples, and is usually given by

$$E_s\left(\mathbf{f}\right) = \sum_{i \in \mathcal{D}_\Lambda} \left(\lambda_i^x \left(f_{i+1} - f_i - d_i^x\right)^2 + \lambda_i^y \left(f_{i+w} - f_i - d_i^y\right)^2\right),$$

for a 5-stencil (Figure 1a)), where d_i^x and d_i^y are the constraint values, and λ_i^x and λ_i^y are again relative-importance-weights. The superscripts ^{x} and ^{y} indicate whether the parameter belongs to a horizontal- or vertical difference constraint. Since the energy functional $E(\mathbf{f})$ is quadratic in the elements of $\mathbf{f}$, the solution to (1) is a LS solution, and can be found by solving a linear equation system of the form $A\mathbf{f} = \mathbf{b}$, where A is a symmetric, quadratic and positive definite matrix. Since the constraints are defined on small, local neighborhoods on Λ, the matrix A is very sparse and only contains a main- and a few off-diagonals. Depending whether the constraints are formulated on a 5-stencil or a 9-stencil (Figure 1a), the problem matrix A has one main-diagonal and four, respectively eight off-diagonals - out of which only half are unique due to symmetry (Figure 1b). Further, the matrix dimension $n = w \times h$ is in the order of tens of thousands to millions - depending on the resolution of Λ. The structure of A in the case of IDW is discussed below.

B. Linear Systems for IDW Applications

We will use an example application called *stereo to multiview conversion* that uses IDW in order to realize small perspective changes of the original shot [6]. This method first extracts image features from a *Stereoscopic-3D* (S3D) video, which are then used to formulate a LS problem. The solution vector $\mathbf{f}$ represents the *coordinates* of the pixels in the transformed image.

The resolution of current video content is predominantly 1080p (1920 pixels wide by 1080 pixels in height), resulting in two (one for each coordinate dimension) to four (when S3D footage is used) equation systems with nearly two million variables for each frame in the video. Solving such large systems at frame rates of up to 30 fps is computationally very demanding and often a pixel dense solution is not required. Therefore, the problems are usually solved on around $10\times$ sub-sampled grids in order to reduce the computational complexity. This results in realistic grid sizes of about 190×110, which equals to $\approx 20\,k$ variables in the minimization problem.

C. Linear Solver Algorithms

Linear solver algorithms fall into two main categories, namely *direct* and *iterative* ones. Direct solvers employ a matrix decomposition (such as LU or CD) in order to compute an exact solution, whereas iterative solvers successively refine an approximate solution. The choice of the solver is dependent on several factors such as the mathematical properties and structure of A, convergence and numerical behaviour, and the complexity of arithmetic operations and memory accesses. In this evaluation, we consider CG- and CD-based solvers since these are two of the most widely used algorithms for positive-definite systems emanating from LS problems. Some of their properties are summarized below. For more details see [7][8][10].

D. Direct CD Solver

The CD comes in two variants, namely the LL^T and the LDL^T decompositions. In both cases, the L matrix is a lower-triangular matrix that can be used to obtain a solution to the equation system $A\mathbf{f} = \mathbf{b}$ with a forward and backward substitution step. The difference is that in the second variant, the diagonal elements of L are offloaded to the diagonal matrix D. We only consider the second variant here, since it is more amenable to hardware implementations [14] due to the absence of square-roots and fewer divisions.

The computational complexity and storage requirements of a CD based solver are in general much higher than for an iteration of CG ($O\left(n^3\right)$ and $O\left(n^2\right)$, respectively). However, the sparsity structure of A leads to a banded Cholesky factor L with bandwidth $bw = h + 1$ or $bw = h{+}2$ (depending on the stencil), where h is the height of the image grid. The complexities then reduce to $O\left(n^2\right)$ and $O\left(n^{1.5}\right)$.

A so called *incomplete factorization* can be used in order reduce the complexity of the decomposition step. The idea is to drop insignificant elements below a certain threshold ρ during the calculation of the large inter-row scalar products. While this scheme is often used in order to calculate pre-conditioners for iterative methods [8], it has also been shown that it can be precise enough to directly compute a solution in the case of the IDW applications at hand [15].

Fill-reducing permutations such as *nested dissection* [20] have not been considered in this paper, since they destroy the regular band structure of the matrix. This impedes scan-line processing of the solver input and outputs, and complicates the solver architecture considerably due to the use of sparse data formats and irregular memory access patterns.

E. Iterative CG Solver

Iterative solvers such as CG start from an initial solution $\mathbf{f}_0$ which is then successively refined according to a specific recurrence equation (shown in Figure 3b for CG). An advantage of iterative solvers is that they can be easily parallelized, and that their memory and computational complexity is in the order of $O\left(n\right)$ for one iteration, where n is the dimension of the matrix A. However, they have to be properly preconditioned in order to ensure rapid convergence, and - due to their iterative nature - they require a very high memory bandwidth. In the CG variant used in this paper, a standard diagonal pre-conditioner $\mathbf{m} = \operatorname{diag}\left(A\right)$ is applied, since this considerably improves the convergence for our application.

IV. Evaluated Hardware Architectures

In this exploration, we consider the following 6 different solver variations: *ldl_std_on* and *ldl_std_off* are essentially standard CD solvers that use either off-chip or on-chip memory to store the factorized matrix and the intermediate result vector $\mathbf{y}$. *ldl_aprx_on* and *ldl_aprx_off* are similar to the *ldl_std* variants, but they perform

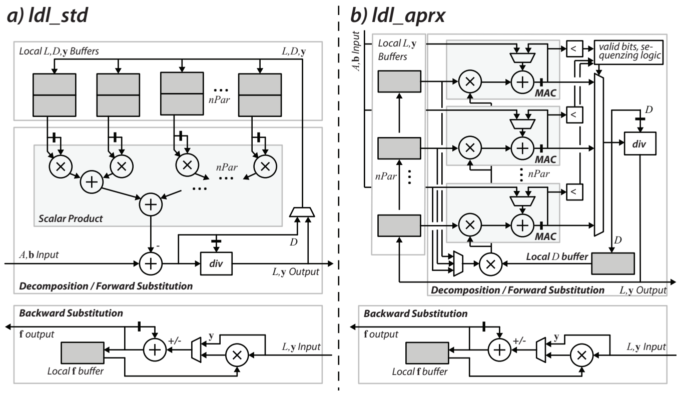

Fig. 2. Datapaths of the two *ldl* variants.

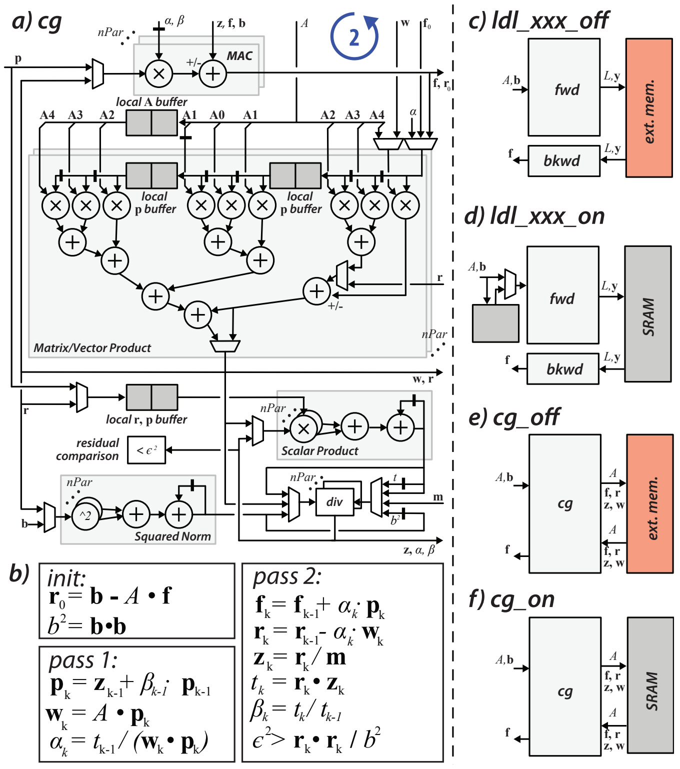
a) cg
b)
c) ldl_xxx_off
d) ldl_xxx_on
e) cg_off
f) cg_on
local A buffer
local p buffer
Matrix/Vector Product
local r, p buffer
residual comparison
Squared Norm
Scalar Product
MAC
div
fwd
bkwd
ext. mem.
SRAM
init: $r_0 = b - A \bullet f$, $b^2 = b \bullet b$
pass 1: $p_k = z_{k-1} + \beta_{k-1} \cdot p_{k-1}$, $w_k = A \bullet p_k$, $\alpha_k = t_{k-1} / (w_k \bullet p_k)$
pass 2: $f_k = f_{k-1} + \alpha_k \cdot p_k$, $r_k = r_{k-1} - \alpha_k \cdot w_k$, $z_k = r_k / m$, $t_k = r_k \bullet z_k$, $\beta_k = t_k / t_{k-1}$, $\epsilon^2 > r_k \bullet r_k / b^2$

Fig. 3. Datapath of the *cg* variants (a), the CG recurrence equations (b), and the different memory configurations of all evaluated solvers (c-f).

an incomplete factorization by skipping L elements with a magnitude lower than a certain threshold ρ when building the inter-row scalar products. The variants *cg_on* and *cg_off*, are $\mathbf{m} = \operatorname{diag}(A)^{-1}$ preconditioned CG solvers that either store their data completely on-chip or off-chip. Each of the solver variants has some additional parameters such as the on-chip memory size or the parallelization degree. The architectures are based on the ones presented in [10][15] and are explained in more detail below.

A. LDL_STD Variants

The architecture of the normal CD solver is shown in Figure 2a, and consists of two parts: The first part performs the decomposition and the forward-substitution (these can be interleaved in the same datapath), and the the second part performs the backward substitution. Although the backward pass is similar to the forward pass, the matrix L and the vector y have to be accessed in reversed order (bottom-up), and therefore this task can only be executed once the decomposition and forward pass have been finished. Using a separate unit is more convenient and allows to perform both the forward- and backward pass of two subsequent matrices in parallel.

The decomposition stage contains an $nPar$ wide scalar-product, which is used to calculate the inter-row products in column-major order. The decomposition is very sequential in nature, since each element L_{ij} depends on all it's neighbours to the left. However, due to the banded shape of L, the values required to compute another column of L all lie within a window of size bw^2. These are buffered locally - together with the past bw elements of the y and D vectors. Parallelization is easy up to a degree of $nPar = bw$. Afterwards, the strong dependencies on previous results impedes further parallelization. Here, the parallelization degree is specified relative to the bandwidth and indicated as postfix in the architecture name. E.g. *ldl_std_off_p/2* corresponds to $nPar = \lceil bw/2 \rceil$.

Two variants of *ldl_std* are considered in this evaluation. The first, *ldl_std_off* stores L and y in the off-chip memory (Figure 3c). The second variant *ldl_std_on* uses only on-chip memory to store L and y. Since L can be very large ($bw \times h \times w$ entries) it may not be feasible to store the whole matrix on-chip. Therefore we also consider variations of *ldl_std_on*, where only a fraction of L is kept in the buffer (the rest is discarded). The solution $\mathbf{f}$ is then computed with several decomposition passes. Consider for example an on-chip buffer which can store half of L. Overall, two decomposition passes are then needed: first, the whole decomposition is calculated, but only the bottom half of L is stored. The bottom half can then be used to perform the backward substitution, while the upper half of the decomposition is recomputed and stored. This variation requires less on-chip memory, but needs more time due to the re-computation steps. Therefore, for all *ldl_std_on*, $nPar$ is set to bw, and the parameter in the postfix denotes the fraction of the L matrix that is kept on-chip. E.g. *ldl_std_on_m/2* is able to store half of L on-chip.

B. LDL_APRX Variants

The *ldl_aprx* variants are in principle similar to *ldl_std*. The difference is that they take advantage of the zeroes in L, and therefore the scalar products cannot be computed in parallel using a rigid multiplier-adder tree structure. Instead, these scalar products are computed sequentially in parallel by individual MAC units (Figure 2b). But the computed L elements are still gathered in column-major order. In the evaluation, the same memory configurations as for *ldl_std* are used (Figure 3c,d).

C. CG Variants

The CG solver iteratively works on the residual vector $\mathbf{r}_k = A\mathbf{f}_k - \mathbf{b}$ (k is the iteration index) and produces several intermediate vectors $\mathbf{r}_k, \mathbf{p}_k, \mathbf{w}_k, \mathbf{x}_k, \mathbf{z}_k$, as shown in Figure 3b. The initial solution $\mathbf{x}_0$ can be arbitrary and is chosen to be the one-vector here. The preconditioner matrix is $\mathbf{m} = \operatorname{diag}(A)^{-1}$. The parameter ϵ denotes the relative residual[2] tolerance and is used as the iteration stop criterion. It's value is determined later in the evaluation.

Both CG variants evaluated in this paper have the same datapath, which is shown in Figure 3c. They only differ in the way they store the intermediate results - i.e. *cg_off* stores everything off-chip whereas *cg_on* stores everything on-chip (Figure 3e,f). The datapath consists of several MAC units for the vector-vector product calculation, several matrix-vector multipliers, and one scalar-product. Since the matrices at hand contain at most 5 or 9 (off-)diagonals, the matrix-vector multipliers can evaluate one row-vector product per cycle. Due to the sequential dependency on the scalar-products α and β, one iteration of CG cannot be evaluated in a single step

[2]The relative residual is defined as $\|\mathbf{r}_k\|/\|\mathbf{b}\| = \|A\mathbf{f}_k - \mathbf{b}\|/\|\mathbf{b}\|$.

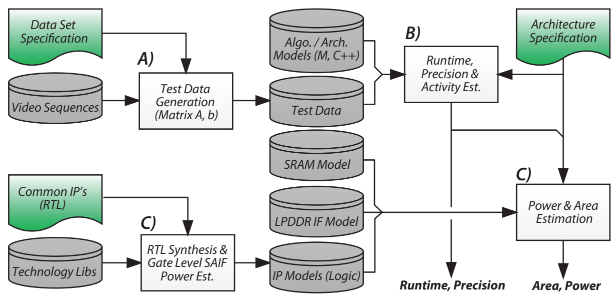

Fig. 4. Flow of the estimation framework used in this work.

and therefore two passes are required. However, the vector-vector, matrix-vector, and scalar-products can be easily parallelized. The amount of parallelization is determined by the parameter $nPar$, and will be used as a postfix in the architecture identifier. I.e. *cg_off_p4* would correspond to a CG solver with off-chip memory and fourfold parallelization $nPar = 4$. Overall, storage for 9 vectors of dimension n has to be provided in the case of a 9 off-diagonal matrix: there are 5 unique diagonals of A and 4 temporary vectors $\mathbf{p}_k$,$\mathbf{f}_k$,$\mathbf{r}_k$ and $\mathbf{w}_k$ ($\mathbf{w}_k$ and $\mathbf{z}_k$ may share the same slot). Memory accesses are completely linear, and hence the utilization of the off-chip memory bandwidth is close to optimal.

V. Estimation Framework

In order to quickly assess different architecture variants and find potential tradeoffs a Matlab based framework has been developed which estimates the *area*, *throughput*, *power consumption* and *precision*. The estimations are based on a mixture of algebraic architecture cost models, pre-characterized logic instances, SRAM datasheet values and a LPDDR3 system power model. It is important to note that the aim is not to achieve very high estimation accuracy, but rather to be able to quickly compare several architecture variants in a simple and efficient manner. In particular, the framework enables early estimations at the system level in order to observe general trends. The estimation flow is illustrated in Figure 4, which contains the three main parts *A) data generation*, *B) runtime, precision and activity estimation* and *C) power and area estimation*- each of which is explained below. Note that this estimation framework could also be used to explore other types of hardware architectures.

A. Test Data Generation

The test data is an integral part of the evaluation since it's statistics directly affect the numerical performance of the evaluated architectures. The test data is generated according to parameters specified in a *Data Set Specification*. This step depends on the application - in this case real-world video sequences area analyzed in order to produce the LS problem matrix A and the vector $\mathbf{b}$. The test data is organized as a batch of sequences which are stored on disk. The architectures to be explored can then be evaluated on these pre-computed batches.

B. Runtime, Precision & Activity

For each of the hardware architectures to be evaluated, a parameterizable multi-precision software model is implemented in order to estimate the quantization error and other statistics such as matrix non-zero patterns and iteration counts. Together with algebraic cost models, these statistics can be used to calculate the throughput of a certain architecture instance, and activities of the individual logic and memory elements can be estimated. The individual parameters of an architecture are defined in an *Architecture Specification*. In fact, such a specification can define several instances, all of which are evaluated on a given test-data batch to make parametric explorations.

C. Power & Area Estimation

The power and area estimations base on the assumption that the dominant contributions (in both circuit area and power consumption) come from the floating point operations (FLOPS), from the on-chip SRAM memories and the off-chip memory subsystem. These three contributions are modeled separately, as explained below.

Logic: The power consumption and area of common floating point operations such as multiplication, addition/subtraction and division are calculated using Synopsys Designware (DW) IP's and a gate-level SAIF back-annotation flow. I.e. the DW IP's are first wrapped in RTL code and synthesized with Synopsys Design Compiler (DC). The obtained netlists are then simulated in Mentor Modelsim and toggle activities are extracted and annotated to a SAIF file, which is then read back into Synopsys Power Analyzer to estimate the active-, leakage- and switching power. In this exploration, post layout parasitics are estimated using wireload models provided by the standard cell library vendor. In order to get a worst case estimate, $2'000$ random testvectors are used in the simulation. These steps are performed for several clock periods and for different parameterizations of the floating point IP's (exponent/mantissa widths). The obtained values are cast into an interpolated look-up table which - together with the runtime and activity values - can then be used to make estimates for arbitrary parameter combinations. For this evaluation, all IP's have been characterized for the TT, 0.8V, 25C corner of a 28 nm CMOS process.

SRAM: In order to estimate circuit area and power consumption of the SRAMs, we extracted the corresponding values from the datasheets of a representative set of high-density, single-port SRAM instances. These values were used to build an interpolated look-up table which enables to make estimates for arbitrarily configured SRAM macros with widths up to 64 bit, and depths up to 16 k. Since 16 k is the maximum depth that the employed memory compiler can generate, larger SRAMs are assembled using several instances.

External Memory: We use an LPDDR3 memory interface for this evaluation, which consists of three parts: *memory controller*, *PHY* and *memory module*. Since the memory architecture is very similar to LPDDR2, the power and area values for the memory controller were taken from [21], where implementation results for a few controller variants in STM 28 nm technology are presented (the results of the largest memory controller are used). These values are then derated to the final operating frequency and scaled to the correct interface width. An estimate for the PHY area is taken from the same source. However, since no power figures for this PHY are provided, they are estimated the LPDDR3 values published by JEDEC [22]. In addition, it is assumed that the data lines are actively terminated to 50 Ω during data transfers (otherwise the termination is switched off). The power consumption of external memory module is based on the *Micron System Power Calculator* for LPDDR2. Since the LPDDR3 spreadsheet is not open-access, we updated it to LPDDR3 using values from the datasheet of the Micron EDF8132A1MC LPDDR3 device. This approach was deemed feasible due to the similarity of LPDDR2 and LPDDR3 and since no special power-down modes exclusive to LPDDR3 are used here. Currently, our external memory model supports the three configurations LPDDR3-400 with 32 bit, LPDDR3-800 with 32 bit and LPDDR3-800 with 64 bit - corresponding to memory bandwidths of 3.2, 6.4 and 12.8 GByte/s.

VI. Evaluation Results

The operating point for all architectures in this evaluation is chosen to be 400 MHz, since the maximum frequency of the employed SRAM

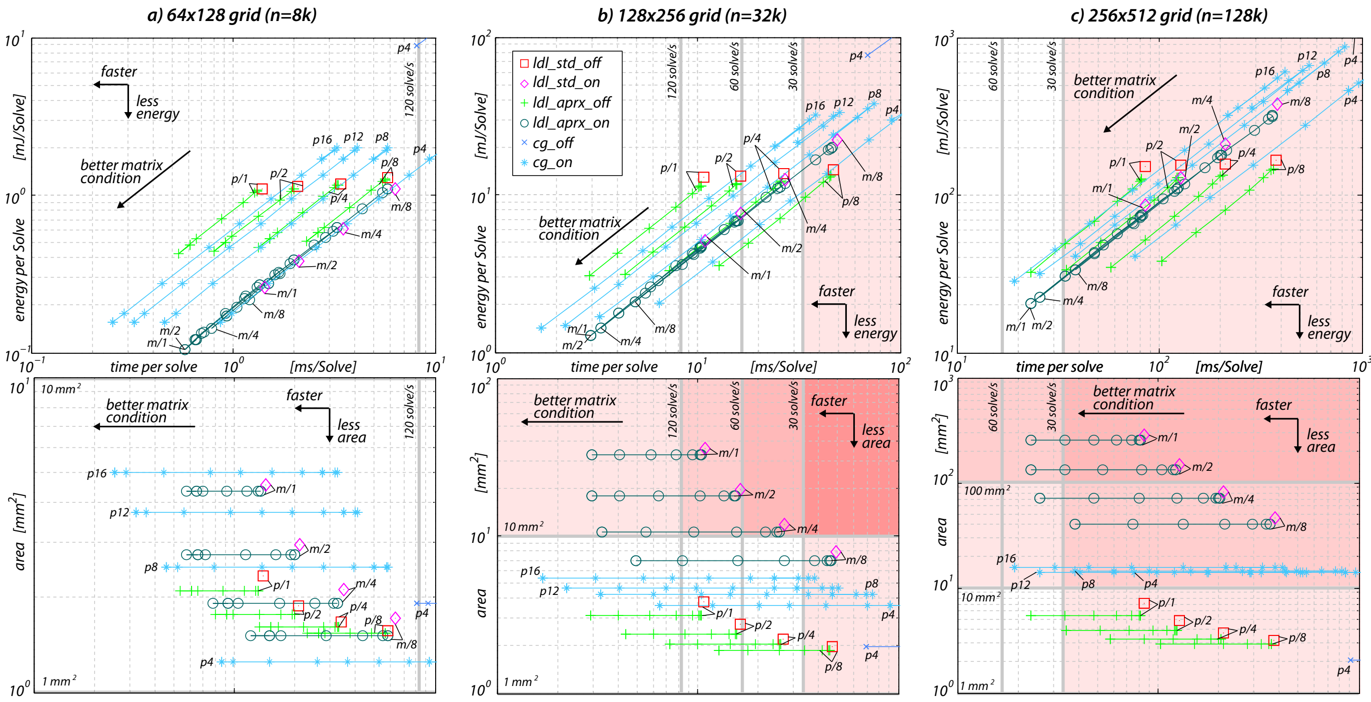

Fig. 5. This figure shows the ATE design space of all evaluated architecture variants for the grid sizes a) 64×128, 128×256 and 256×512. Areas which are deemed infeasible due to an insufficient throughput or a very large area requirement are shaded with red.

macros lies around 450 MHz. Further, all off-chip variants have a LPDDR3-800 interface with a maximum bandwidth of 6.4 GByte/s, which is a reasonable configuration for mobile devices. For all architectures except the *ldl_aprx* variants, an additional logic overhead of 10% is assumed. In the case of *ldl_aprx*, a higher value of 20% is applied to account for additional logic coming from the two large multiplexers and zero skipping logic. We base this overhead on our practical experiences gained during the design and implementation of several such systems in both ASICs and FPGAs.

In this evaluation, we use IDW problem matrices emanating from automatic multiview synthesis. The results represent averages over ten frames of ten S3D video sequences[3]. The evaluated matrix sizes are $n = \{8\,\mathrm{k}, 32\,\mathrm{k}, 128\,\mathrm{k}\}$, and correspond to LS problems defined on $h \times w = 64 \times 128$, 128×256 and 256×512 grids using a 9-stencil. Precision related issues are discussed next, followed by a discussion of the main results.

A. Matrix Condition, Arithmetic Precision and Solution Accuracy

The matrix condition considerably influences the convergence properties and solution accuracy of all solvers [8]. In the problem at hand, the condition is mainly influenced by the amount of data constraints in (1) [15]. The more data constraints are available, the lower is the condition number κ, and the better is the solvability. Intuitively, this is because the LS problem propagates the data constraint values from the constraint positions to adjacent variables via the differential smoothness constraints. The less data constraints are given, the further these values have to be propagated and the more susceptible the solution process is to quantization errors. Therefore, the dataset used in this evaluation comprises matrices with low to very high condition numbers. The arithmetic precision (number of mantissa bits) and additional parameters such as the tolerance ϵ of the CG solver, and the threshold ρ of the incomplete CD are determined for each matrix size separately using the worst-conditioned problem in the dataset. Note that that the solution $\mathbf{f}$ represents pixel coordinates and therefore it is sufficient for the maximum absolute- and median absolute error to lie below 1 and 0.1 pixel, respectively (smaller errors are imperceptible).

[3]www.rmit3dv.com

B. Results

The ATE design space of all evaluated variants is shown in Figure 5, and Figure 6 shows additional information such as the external bandwidth, and a more detailed energy- and area split in the case of 128×256 matrices. We can see that the *ldl_std* variants are suboptimal in all cases. The *cg_off* variant has very low area requirements, but apart from this, it is quite unattractive from an energy and throughput point of view. This is due to the many off-chip memory accesses, i.e. in Figure 6a) it can be seen that it is completely memory bound. The *ldl_aprx_on* variants can reach very high energy efficiency comparable to the on-chip CG solvers, but they are not very attractive in terms of scalability. A CG based solver which uses only on-chip resources can provide a very good energy efficiency at moderate area consumption in all cases - provided that the matrix is very well-conditioned. A similar observation can be made for the *ldl_aprx_off* variants, although they do not reach the same level of energy efficiency as *cg_on*. They are attractive for grid sizes is in the order of 128×256 to 256×512 if circuit area is of utmost importance, and if a memory interface with sufficient off-chip bandwidth (Figure 6a) is available.

C. Discussion

There are two main usage scenarios for the accelerators at hand: either they are integrated in a separate chip, or they are integrated as a part of an SoC. In both scenarios, a DRAM free solution is highly desirable. In the first case, the chip would have much simpler IOs and packaging, and it minimizes system cost and board level design effort. In the SoC scenario, a DRAM controller is probably available, but DRAM bandwidth will be in high demand due other subsystems (CPU, GPU, DSP...). So adding an additional DRAM-bandwidth hungry accelerator is a difficult proposition. As can be seen in Figure 6a, all off-chip variants use a significant percentage of

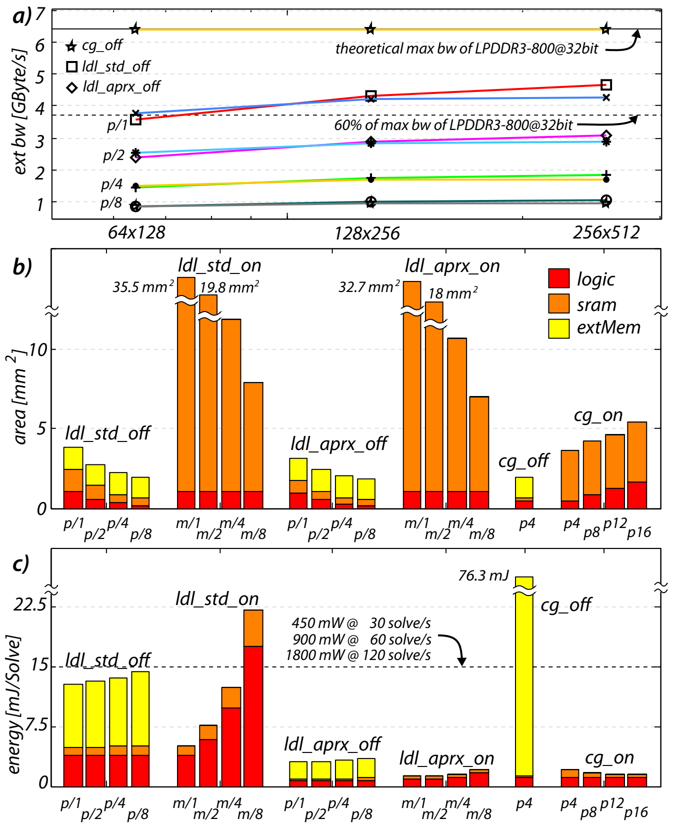

Fig. 6. a) External bandwidth for all off-chip variants (well-conditioned case). 100% and 60% of the maximum theoretical bandwidth are indicated with a solid black and a dashed black line. The latter is a more realistic estimate for the maximum bandwidth available in a real system.In b) and c) an area and energy split is shown for 128×256 grids (well-conditioned case). For comparison, the GPU subsystems of Apples A7 and NVIDIAs Kepler are estimated to occupy around 21 mm^2 and 60 mm^2, respectively[5].

the available external bandwidth. An accelerator that fully works in a *streaming*-like manner is much easier to insert at the SoC system level - like the image signal processor (ISP) which are used to process the camera output in modern SoCs.

In this perspective, the iterative *cg_on* solver variant is highly attractive since it does not require as much silicon area as the *ldl_on* variants, and since it is easier parallelizable - which is important to support larger matrix sizes in the future. Issues due to the matrix condition are fortunately less severe than the results might indicate, since the matrices are often well-conditioned in video processing applications such as the one at hand. This is because the problems to be solved usually contain a temporal consistency component (i.e. the current frame shall be similar to the last frame), which manifests itself in many data constraints [15]. Further, the iterative CG solver can make use of additional improvements such as using the previous solution as an initial guess, multi-level techniques [10] and more elaborate pre-conditioners [2].

VII. Conclusions

We have seen that if circuit area is the highest priority, an incomplete CD based solver can be an attractive solution. But if enough silicon area is available, iterative solvers such as CG provide the best tradeoff at the system level in this setting - given that pre-conditioning issues are properly tackled. In addition, their parallelization is almost trivial as opposed to direct solver variants. Today, a on-chip CG solver for 128 k matrices is completely feasible. Larger matrix sizes will soon be feasible as well, since it is expected that next generation technologies will provide much more on-chip memory due to the combination of higher integration densities and 3D-stacking techniques.

Full ASIC implementations of the most promising variants are planned in order to verify the estimation results. Further, we have seen that approximation techniques such as the incomplete CD can boost the performance of direct solvers, and therefore it would also be interesting to look into approximation concepts for iterative solvers - possibly on the micro-architectural- or on the circuit level [23] - since these solvers are self-correcting to some degree.

[5]`www.fool.com/investing/general/2014/06/04/just-how-big-is-nvidia-corporations-tegra-k1.aspx`

Acknowledgements

We would like to thank Denis Dutoit (CEA LETI) for his advice on modelling the external RAM interface. This work was partially supported by the FP7 ERC Advance project MULTITHERMAN (g.a. 291125) and by the YINS RTD project (no. 20NA21 150939), evaluated by the Swiss NSF and funded by Nano-Tera.ch with Swiss Confederation financing.

References

[1] C. Kerl, J. Sturm, and D. Cremers, "Dense visual slam for rgb-d cameras," in *IEEE IROS 2013*, Nov 2013.
[2] D. Krishnan and R. Szeliski, "Multigrid and multilevel preconditioners for computational photography," in *ACM TOG*, vol. 30, no. 6, 2011.
[3] I. Koutis et. al., "Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing," *Computer Vision and Image Understanding*, vol. 115, no. 12, 2011.
[4] P. Krähenbühl et. al., "A System For Retargeting of Streaming Video," *ACM ToG*, vol. 28, no. 5, Dec. 2009.
[5] Lang et. al., "Nonlinear Disparity Mapping for Stereoscopic 3D," *ACM ToG*, vol. 29, no. 4, July 2010.
[6] N. Stefanoski et. al., "Automatic view synthesis by image-domain-warping," *TIP*, vol. 22, no. 9, Sept 2013.
[7] A. Björck, *Numerical Methods for Least Squares Problems*. SIAM'96.
[8] Y. Saad, "Iterative Methods for Sparse Linear Systems Second Edition," *SIAM*, 2003.
[9] A. Roldao and G. A. Constantinides, "A high throughput fpga-based floating point conjugate gradient implementation for dense matrices," *ACM TRETS*, vol. 3, no. 1, Jan. 2010.
[10] P. Greisen et. al., "Evaluation and FPGA Implementation of Sparse Linear Solvers for Video Processing Applications," *TCSVT*, Aug. 2013.
[11] O. Maslennikow et. al., "Parallel implementation of Cholesky LL^T-Algorithm in FPGA-based processor," in *PPAM*, 2008.
[12] H. Cho, J. Lee, and Y. Kim, "Efficient Implementation of Linear System Solution Block Using LDL^T Factorization," *SoC 2008*, vol. 03, 2008.
[13] J. Sun, G. Peterson, and O. Storaasli, "High-performance Mixed-Precision Linear Solver for FPGAs," *IEEE TC*, vol. 57, no. 12, 2008.
[14] Y. Depeng et. al., "Compressed Sensing and Cholesky Decomposition on FPGAs and GPUs ," *Parallel Computing*, vol. 38, no. 8, 2012.
[15] M. Schaffner et. al., "An approximate computing technique for reducing the complexity of a direct-solver for sparse linear systems in real-time video processing," in *DAC*, 2014.
[16] U. Vishnoi and T. G. Noll, "Cross-layer optimization of QRD accelerators," in *ESSCIRC 2013*, 2013.
[17] S. Thoziyoor et. al., "Cacti 5.1," *HP Laboratories, April*, vol. 2, 2008.
[18] Sheng Li et. al., "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO-42*.
[19] M. Lang et. al., "Practical Temporal Consistency for Image-based Graphics Applications," *ACM ToG*, 2012.
[20] A. George, "Nested dissection of a regular finite element mesh," *SIAM Journal on Numerical Analysis*, 1973.
[21] B. Boroujerdian et. al., "LPDDR2 Memory Controller Design in a 28 nm Process," 2012, www.eecs.berkeley.edu/~bkeller/rekall.pdf.
[22] F. Ross, "A review of lpddr3 commands, operations & functions," 2012, www.jedec.org/lpddr3-presentations.
[23] K. Palem and A. Lingamneni, "Ten Years of Building Broken Chips: The Physics and Engineering of Inexact Computing," *ACM TECS*, 2013.