Exploiting Three-Dimensional (3D) Memory Stacking to Improve Image Data Access Efficiency for Motion Estimation Accelerators

Yiran Li[a], Yang Liu[b], Tong Zhang[a]

[a] *ECSE Department, Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY 12180, USA*
[b] *Juniper Networks, 1194 North Mathilda Avenue, Sunnyvale, CA 94089, USA*

Abstract

Enabled by the emerging three-dimensional (3D) integration technologies, 3D integrated computing platforms that stack high-density DRAM die(s) with a logic circuit die appear to be attractive for memory-hungry applications such as multimedia signal processing. This paper considers the design of motion estimation accelerator under a 3D logic-DRAM integrated heterogeneous multi-core system framework. In this work, we develop one specific DRAM organization and image frame storage strategy geared to motion estimation. This design strategy can seamlessly support various motion estimation algorithms and variable block size with high energy efficiency. With a DRAM performance modeling/estimation tool and ASIC design at 65nm, we demonstrate the energy efficiency of such 3D integrated motion estimation accelerators with a case study on HDTV multi-frame motion estimation.

Key words: Motion estimation (ME), 3D memory stacking

1. Introduction

As the most resource demanding operation in video encoding, motion estimation has been widely studied over the past two decades and many motion estimation algorithms have been developed, covering a wide spectrum of trade-offs between motion estimation accuracy and computational complexity. Most existing algorithms perform block-based motion estimation, where the basic operations are calculation and comparison of the matching functions between current image macroblock (MB) and all the candidate MBs inside a confined area in reference frame(s). The most common matching function is the sum of absolute differences (SAD). In current design practice, the entire current and reference frames are typically stored in large-capacity off-chip commodity DRAM, and logic chips that implement video coding typically contain on-chip SRAM as buffers to streamline the operations and reduce the off-chip DRAM access frequency. Such on-chip SRAM tends to occupy a relatively large amount of silicon area, e.g., more than half of the video codec area is occupied by SRAM in the design reported in [1], and most on-chip SRAM is used for motion estimation. As video image resolution and frame rate continue to increase and multi-frame based motion estimation is being widely used, image data access tends to account for an increasingly significant percentage of overall video coding energy consumption.

Although tremendous research efforts have been devoted to energy-efficient realizations of motion estimation, under the current design practice where motion estimation computation and image frame storage locate on separate chips, the achievable energy efficiency is subject to a fundamental limit: Under the logic die size constraint in practice, the on-chip SRAM buffer can only hold a small portion of the reference image(s). As a result, we still need relatively frequent access to the off-chip large-capacity commodity DRAM and the same reference image may be fetched for several times. Hence, the off-chip commodity DRAM access tends to consume a significant amount of energy, regardless to how the motion estimation algorithms are being designed and optimized. Moreover, to maximize the energy efficiency of many systems, particularly portable devices, it is highly desirable to improve the power awareness. This demands the motion estimation engine should be able to support algorithms with different energy consumption and performance trade-offs. However, because memory data access tends to account for a significant percentage of overall energy consumption, the separation of frame storage and motion estimation computation in current design practice tends to make it difficult to seamlessly support such dynamic configuration for the realization of graceful power awareness.

Because of significant recent developments of three-dimensional (3D) integration technology with massive vertical interconnect bandwidth towards high manufacturability [2], 3D logic-memory integrated processing platform, which consists of one logic die and one or more high-density memory (such as DRAM) die(s), appears to be very attractive for memory-hungry applications. In general, 3D integration refers to a variety of technologies which provide electrical connectivity between multiple active device planes. Monolithic, wafer-level, back-end-of-the-line

(BEOL) compatible 3D integration, which is enabled by wafer alignment, bonding, thinning and inter-wafer interconnections with through silicon vias (TSVs), appears to be the most economically and technologically viable option for mass-product 3D integration [2]. Hence, this work considers the use of such wafer-level TSV-based 3D integration. By providing massive interconnect bandwidth between logic and memory dies with very short distance, 3D logic-memory integration can most effectively enable the heterogeneous multi-core platform for future converged computing and multimedia processing systems. It should be pointed out that the 3D stacked memory does not intend to replace off-chip large-capacity commodity DRAM. This is mainly because 3D stacked memory must have the same die size as the logic die, which may be much smaller than the die size of off-chip commodity memory. Moreover, to serve for various cores and accelerators, the 3D stacked memory should have a highly distributed structure with many I/O interfaces, which tends to result in a reduced storage density. Therefore, 3D stacked memory in our interested 3D logic-memory integration may not be able to satisfy the overall storage capacity required by the entire heterogeneous multi-core system. In fact, the 3D stacked memory can be considered as an in-package cache/buffer for the off-chip commodity memory, e.g., 3D stacked memory may serve as a large-capacity last-level cache for general-purpose processor cores.

Under such a 3D logic-memory integrated heterogeneous multi-core platform, this work concerns the design of energy-efficient and highly versatile motion estimation accelerators. Because of the high density and low cost advantages of DRAM, we are particularly interested in 3D logic-DRAM integration. Intuitively, we can use the 3D stacked DRAM to store a few complete image frames required for present motion estimation operations, which clearly can convert a majority of off-chip commodity DRAM data access to 3D stacked DRAM data access. Because 3D stacked DRAM data access can be more energy efficient, it is reasonable to expect improvement of motion estimation energy efficiency. Moreover, the use of 3D stacking provides a large room for DRAM structure customization and optimization geared to target applications. In this work, we develop one specific DRAM architecture design and corresponding image frame storage strategy geared to motion estimation, which can effectively leverage the high logic-DRAM interconnect bandwidth enabled by 3D integration, and meanwhile seamlessly support various motion estimation algorithms and hence enable a graceful power-awareness. Moreover, we develop a bit-plane DRAM organization strategy that can naturally support on-the-fly configuration of the trade-off between image pixel precision and 3D DRAM data access power consumption. We note that, in the presence of such customized 3D DRAM for image storage, we can either completely eliminate on-chip SRAM buffer to reduce the logic die area or still keep on-chip SRAM buffer as in current design practice to further reduce energy consumption. We carry out detailed 3D DRAM modeling and ASIC design at 65nm node and consider various design scenarios out to quantitatively demonstrate the energy efficiency of such 3D DRAM based motion estimation and study the involved trade-offs.

2. Proposed Design Strategies

This work concerns how to leverage 3D DRAM stacking to improve the performance of video coding systems, in particular motion estimation, under a 3D logic-DRAM integrated heterogeneous multi-core system as illustrated in Fig. 1. It contains a heterogeneous multi-core logic die and multiple DRAM dies that provide large data storage capacity and very high data access bandwidth for the heterogeneous multi-core systems. Those common heavy duty functions can be realized as application-specific accelerators to improve the overall system energy efficiency.

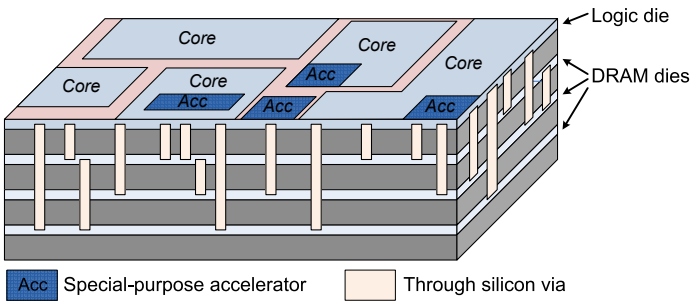


Figure 1: Illustration of 3D logic-DRAM integrated heterogeneous multi-core computing system.

In current design practice, the logic chip that implements video coding fetches image frames from an off-chip commodity DRAM through high-speed chip-to-chip data links such as DDR3. As image resolution keeps increasing, such chip-to-chip data links must provide a higher bandwidth and hence tend to consume more energy. In the most straightforward manner, the use of 3D stacked DRAM can largely reduce the load on off-chip data links and hence reduce the associated energy consumption, without demanding any changes on the design of both the video codec and DRAM. As discussed in [3, 4], for chip-to-chip data links, power is mainly dissipated on link termination resistors and drivers. In the context of 3D integration, the much shorter signal path within the range of tens of μm completely obviates the drivers and terminations. Since the capacitive load of TSV (approximately 38 fF) is much lower compared with that of chip-to-chip bus, CMOS buffers can be simply used to connect DRAM dies and logic die. Therefore, as demonstrated in [4], a typical DDR3 data link with an effective dynamic on-die termination (ODT) resistance of 25Ω dissipates 15.9mW, while its counterpart in 3D integration only consumes 0.38mW, representing about 40$\times$ power reduction. Meanwhile, it is clear that the DRAM data fetch latency will also largely reduce in the 3D integration scenario.

Beyond being used to reduce image fetch power consumption as discussed above, 3D DRAM stacking can be further exploited to improve overall system performance if we could rethink how to architect the video codec and DRAM. It is well known that motion estimation is the most resource demanding operation in video coding and largely determines the overall system performance. Hence, this work focuses on the design of DRAM architecture for implementing motion estimation accelerators. As the key operation of motion estimation, SAD computation involves two tasks, including image data retrieval and arithmetic computation. The essential objective of motion estimation accelerator design is to most effectively carry out the SAD computations according to certain motion vector search rules as specified by different motion estimation algorithms. In this section, aiming at exploiting 3D DRAM stacking, we present one DRAM organization and image data storage strategy.

2.1. DRAM Architecture for Image Frame Storage

It is feasible and certainly preferable to store both current and a few corresponding reference frames in 3D stacked DRAM. Due to the access latency and energy consumption overhead induced by DRAM row activation, it is always favorable to make as many consecutive reads/writes as possible on the same row before switching to another row. Therefore, if image blocks are linearly row-by-row (or column-by-column) mapped to the physical address space, a significant amount of row activations will occur, leading to high access latency and energy consumption. It is very straightforward that, instead of such a linear translated mapping, a MB-by-MB mapping geared to motion estimation should be used [5], i.e., each row stores the luminance intensity data of one or more MBs.

Usually a search region spans several MBs in reference frames. Let each MB be $N \times N$, and S_W and S_H represent the width and height of the search region, each search region spans $S_{BW} \times S_{BH}$ MBs in the reference frame, where

$$S_{BW} = 2 \cdot \left\lceil \frac{S_W - N}{2 \cdot N} \right\rceil + 1,$$
$$S_{BH} = 2 \cdot \left\lceil \frac{S_H - N}{2 \cdot N} \right\rceil + 1.$$

In this work, we are mainly interested in the scenario where the 3D stacked DRAM delivers a candidate MB row-by-row to the logic die, i.e., during each clock cycle, the 3D stacked DRAM delivers the luminance intensity data of one row within a candidate MB to the logic die. This is referred to as *per-row DRAM-to-logic data delivery*.

In current design practice, a DRAM chip usually consists of multiple banks, which can be accessed independently, in order to improve the data access parallelism. Since each candidate MB at most spans 2×2 MBs within the search region and each row in a candidate MB at most spans two MBs in reference frame, we store each image frame in two banks that alternatively store all the MBs row-by-row. For example, suppose each MB is 16×16 and the search region is 32×32, then we have $S_{BW} = S_{HW} = 3$, as illustrated in Fig. 2, where the numbers (i.e., either 0 or 1) within the nine MBs indicate the index of the two DRAM banks.

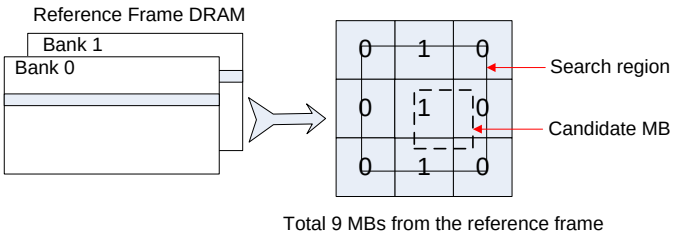


Figure 2: Frame memory organization for row-by-row data delivery when $S_{BW} = S_{HW} = 3$.

Given the current MB and motion vector, the candidate MB from the search region can be readily retrieved row-by-row. Since one DRAM word-line contains multiple consecutive rows within the same search region, once each DRAM word-line is activated and the data of the entire word-line are latched in the sense amplifiers, multiple rows within the search region can be sent to the logic die before we switch to another word-line. Assume luminance intensity of every pixel is represented by D bits, as shown in Fig. 3, each clock cycle two DRAM banks delivers two N-pixel rows within the search region, which are further shifted and combined to form one row in the candidate MB according to the present motion vector.

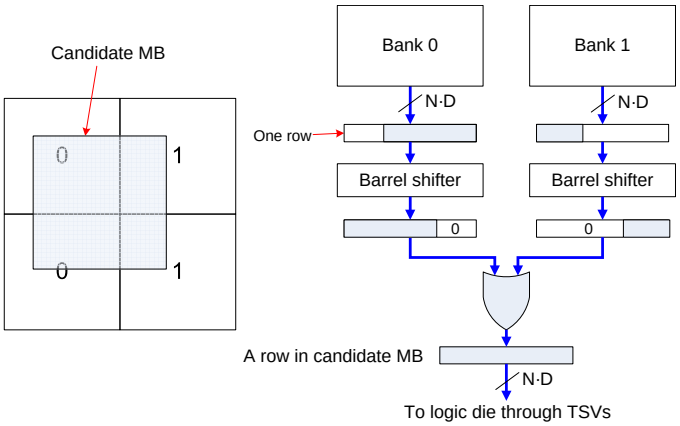


Figure 3: Combine the data from two banks to form a row in candidate MB.

We note that, if an extremely high memory access throughput is required for certain applications, it is possible to deliver the luminance intensity of an entire candidate MB to logic die each clock cycle, which is referred as *per-MB DRAM-to-logic data delivery*. In order to deliver one candidate MB every clock cycle, each reference image frame should be stored in 4 different banks, because each candidate MB at most spans 4 MBs in the reference frame. Based upon current MB location and motion

vector, one corresponding word-line in each bank is activated and delivers the luminance intensity of one MB at once. All the four MBs from the four DRAM banks are sent to a two-dimensional barrel shifter array in order to form the candidate MB. This per-MB data delivery strategy can achieve very high throughput, but obviously will demand a large amount of TSVs and incur much higher energy consumption. Therefore, we only focus on the per-row DRAM-to-logic data delivery strategy throughout the remainder of this paper.

We note that since each row contains several MBs and each MB may contain a few hundred bits (e.g., given $N = 16$ and $D = 8$, each MB contains 2048 bits), a row within a bank should be further partitioned into several segments in order to maintain a reasonable access speed. Therefore, we follow the conventional DRAM design practice that employs a bank→sub-bank→sub-array hierarchy. Each DRAM bank consists of several identical sub-banks and at one time only one sub-bank can be accessed. By segmenting each row within one sub-bank, each sub-bank is further partitioned into a certain number of sub-arrays that collectively carry out DRAM access, and each sub-array contains an indivisible array of DRAM cells surrounded by supporting peripheral circuits such as address decoders/drivers, sense amplifiers and output drivers etc.

To map the MB luminance intensity data onto the sub-arrays within each sub-bank, we propose a *bit-plane* data mapping strategy to ensure the memory data access regularity across all the sub-arrays. Recall that the luminance intensity of one pixel uses D bits. We partition each sub-bank into D sub-arrays, $A_0, A_1, ..., A_{D-1}$, and each sub-array A_i stores the i-th bit of the luminance intensity data, as depicted in Fig. 4. Clearly, this bit-plane storage strategy ensures that all the sub-arrays within the same sub-bank can readily share the same row and column address and have a minimum sub-array data I/O routing overhead. In particular, each sub-array only needs to send N bits for realizing per-row DRAM-to-logic data delivery. In contrast, if we store the D bits for each pixel consecutively in the same sub-array, in stead of using such bit-plane storage strategy, different sub-arrays may contribute to the candidate MB data differently. As a result, the sub-arrays within the same sub-bank may not be able to fully share all the row and column address and each sub-array must have a data I/O width of $N \cdot D$ bits.

Another advantage of this proposed bit-plane storage strategy is that it can easily support run-time graceful performance vs. energy trade-off, because the bit-plane structure makes it very easy to adjust the precision of luminance intensity data participating in motion estimation. It is well-known that appropriate pixel truncation [6] can lead to substantial reduction on computational complexity and power consumption without significantly affecting image quality. Such bit-plane memory structure can naturally support dynamic pixel truncation, which can meanwhile reduce the power consumption of memory data access. Given the D-bit full precision of luminance intensity

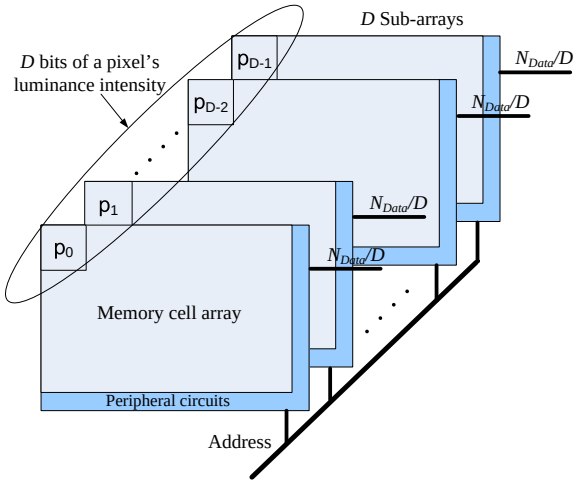


Figure 4: Illustration of *bit-plane* data mapping strategy. All D sub-arrays share the same row and column address. N_{data}-bit data bus is uniformly distributed across D sub-arrays, where $N_{data} = N \cdot D$.

data, if we only use $D_r < D$ bits in motion estimation, we can directly switch the $D - D_r$ sub-arrays, which store the lower $D - D_r$ bits for each pixel, into an idle mode to reduce the DRAM power consumption. It is intuitive that such lower-precision operation can be dynamically adjusted to allow more flexible performance vs. energy trade-off, e.g., we could first use low-precision data to calculate coarse SADs, and then run block matching with full precision in a small region around the candidate MB with the least coarse SAD.

It should be pointed out that, unlike conventional design solutions, the above presented design strategy under the 3D logic-DRAM integrated system framework can realize any arbitrary and discontinuous motion vector search and hence seamlessly support most existing motion estimation algorithms. Finally, we note that, although the above discussion only focuses on data storage for motion estimation, the same DRAM storage approaches can be used to facilitate the motion compensation as well in both video encoders and decoders.

2.2. Logic and Memory Interface

With the above presented DRAM architecture design strategies, the motion estimation engine on the logic die can access the 3D DRAM to efficiently fetch the current MB and candidate MB through a simple interface. Assume the video encoder supports a multi-frame motion estimation with up-to m reference frames. In order to seamlessly support multi-frame motion estimation while maintaining the same video encoding throughput, we store all these m reference frames separately, each reference frame is stored in 2 banks when using per-row DRAM-to-logic data delivery. The motion estimation engine can access all the m reference frames simultaneously, i.e., the motion estimation engine contains m parallel units, each one carries out motion estimation based upon one reference frame. We

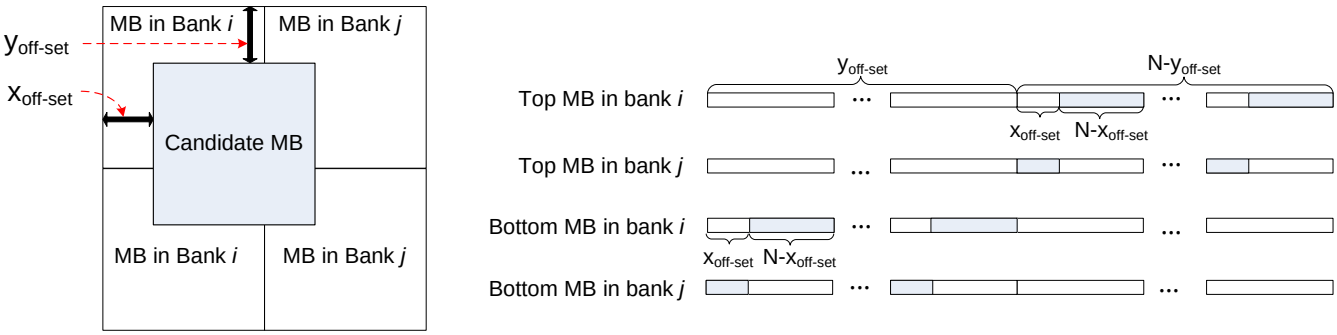


Figure 5: Illustration of calculating memory access address to fetch one candidate MB.

assign each MB with a two-dimensional index with the origin at the top-left corner of each image, i.e., assuming each frame contains $F_W \times F_H$ MBs, the MBs at the top-left corner and bottom-right corner of each frame have the index of (0, 0) and $(F_W - 1, F_H - 1)$, respectively. Assume each word-line in one bank stores s MBs, and we store each MB row-by-row along a DRAM word-line. Hence, given the MB index (x, y), we first can identify its bank index and the DRAM row/column address for proposed DRAM architectures as listed in Table 1. Since each address location in one bank corresponds to one row in one MB and each MB has N rows, each MB spans N consecutive column addresses. The first column address is A_{col} in the table, and the N consecutive column addresses are $\{A_{col}, A_{col} + 1, \cdots, A_{col} + N - 1\}$.

Table 1: Memory access address for proposed DRAM architectures.

Bank Index	$x\%2$
Row Address	$\left\lfloor \frac{y \cdot F_W + x}{2 \cdot s} \right\rfloor$
Column Address (A_{col})	$((y \cdot F_W + x)\%s) \cdot N$

For each motion estimation unit on the logic die to access one reference frame, it only needs to provide two sets of parameters, 2D position index of current MB (x_{MB}, y_{MB}) and motion vector (x_{MV}, y_{MV}), which determine the location of the candidate MB in the reference frame. Given these inputs, DRAM will deliver the corresponding candidate MB to the logic die. Leveraging the specific DRAM architecture described above, each DRAM frame storage can easily derive its own internal row and column address and the corresponding configuration parameters for the barrel shifters, which is described as follows.

As pointed out earlier, each candidate MB may span at most 4 adjacent MBs in the reference frame as illustrated in Fig. 3. Given the 2D position index of current MB (x_{MB}, y_{MB}) and motion vector (x_{MV}, y_{MV}), we can directly derive the 2D position indices of these up-to 4 adjacent MBs as

$$\left(x_{MB} + s_x \cdot \left\lceil \frac{|x_{MV}|}{N} \right\rceil,\ y_{MB} + s_y \cdot \left\lceil \frac{|y_{MV}|}{N} \right\rceil\right),$$
$$\left(x_{MB} + s_x \cdot \left\lceil \frac{|x_{MV}|}{N} \right\rceil,\ y_{MB} + s_y \cdot \left\lfloor \frac{|y_{MV}|}{N} \right\rfloor\right),$$
$$\left(x_{MB} + s_x \cdot \left\lfloor \frac{|x_{MV}|}{N} \right\rfloor,\ y_{MB} + s_y \cdot \left\lceil \frac{|y_{MV}|}{N} \right\rceil\right),$$
$$\left(x_{MB} + s_x \cdot \left\lfloor \frac{|x_{MV}|}{N} \right\rfloor,\ y_{MB} + s_y \cdot \left\lfloor \frac{|y_{MV}|}{N} \right\rfloor\right),$$

where $s_x = \text{sign}(x_{MV})$ and $s_y = \text{sign}(y_{MV})$, and $\text{sign}(\cdot)$ represents the sign of the operand. Clearly, dependent upon the specific values of x_{MV} and y_{MV}, we may have 1, 2, or 4 distinctive indices, i.e., the candidate MB may span 1, 2, or 4 adjacent MBs in the reference frame. For the general case, let us consider the scenario when one candidate MB spans 4 adjacent MBs in the reference frame as illustrated in Fig. 5, where $i \neq j$ and $i, j \in \{0, 1\}$.

After obtaining the indices of the 4 reference MBs, we can directly calculate the corresponding DRAM row addresses. In order to determine the range of the column addresses in the case of per-row DRAM-to-logic data delivery, we should calculate

$$y_{off\text{-}set} = \begin{cases} |y_{MV}|\%N, & y_{MV} \geq 0 \\ N - |y_{MV}|\%N, & y_{MV} < 0 \end{cases}. \quad (1)$$

The candidate MB spans the last $N - y_{off\text{-}set}$ rows in the two top MBs and the first $y_{off\text{-}set}$ rows in the two bottom MBs, as illustrated in Fig. 5. Hence, we can correspondingly determine the column addresses. Then we should determine the configuration data for the barrel shifters that form one row in the candidate MB, i.e., we should calculate

$$x_{off\text{-}set} = \begin{cases} |x_{MV}|\%N, & x_{MV} \geq 0 \\ N - |x_{MV}|\%N, & x_{MV} < 0 \end{cases}. \quad (2)$$

For each row read from the reference MBs on the left as shown in Fig. 5, the barrel shifters shift the data towards left by $x_{off\text{-}set}$ pixels; for each row read from the reference MBs on the right as shown in Fig. 5, the barrel

Table 2: Modeling results for one 2M-byte 2-bank 3D stacked DRAM in the case of per-row DRAM-to-logic data delivery.

	1-layer stacking			
Number of sub-banks	1	2	4	8
Access time (non-burst) (ns)	9.49	7.37	6.65	6.57
Burst access time (ns)	7.17	4.83	4.11	3.91
Energy per access (non-burst) (nJ)	0.71	1.00	1.29	1.61
Energy per burst access (nJ)	0.15	0.45	0.76	1.08
Footprint (mm^2)	1.10	2.08	3.12	4.79
	4-layer stacking			
Number of sub-banks	1	2	4	8
Access time (non-burst) (ns)	9.36	7.25	6.53	6.44
Burst access time (ns)	7.09	4.84	4.03	3.83
Energy per access (non-burst) (nJ)	0.93	1.22	1.28	1.59
Energy per burst access (nJ)	0.13	0.43	0.74	1.06
Footprint (mm^2)	0.22	0.34	0.47	0.69

shifters shift the data towards right by $N - x_{off\text{-}set}$ pixels. By implementing the above simple calculations in the DRAM domain, the 3D stacked DRAM can readily deliver the candidate MB to the motion estimation engine on the logic die, while the motion estimation engine only needs to supply the current motion vector and the position of the current MB.

3. Simulation Results

This section presents our case study on multi-frame motion estimation to demonstrate the above presented design approach. In particular, we set the following parameters. The size of each MB is 16×16 and search region is 80×80 for HDTV1080p (1920×1080) with a frame rate of 30 frames per second. We assume the luminance intensity of each pixel uses 8 bits, hence each DRAM sub-bank has 8 sub-arrays. We assume up-to 5 reference frames can be used during video encoding. Therefore, to access data in each frame simultaneously using the per-row DRAM-to-logic data delivery strategy, the 3D stacked DRAM has $2 \times 6 = 12$ banks and an aggregate data I/O width of $16 \times 8 \times 6 = 768$ bits (i.e., 768 TSVs). For DRAM performance modeling and energy estimation, we use the 3D DRAM design strategy presented in [7] that has been implemented based upon the widely used memory modeling tool CACTI [8]. All the DRAM modeling is carried out at the 65nm technology node.

When using the per-row DRAM-to-logic data delivery, each frame is stored in two banks and each bank has a 128-bit data I/O width. Although each two-bank DRAM module can store multiple frames, in this work we only consider the storage of a single frame with 2M-byte. We explore the 3D DRAM design space by varying the size of each sub-array and the number of sub-banks. In this study, the number of bit-lines in each sub-bank is fixed as 512. Table 2 shows the parameters of 3D stacked DRAM, where both 1-layer and 4-layer DRAM die stacking are considered.

In DRAM, the access time is the time to read a random bit from a pre-charged sub-bank, while it takes less time to read data along the same word-line (i.e., during the burst mode of DRAM access) as shown in Table 2, because it does not require additional time for word-line address decoding and activations, etc. Clearly, the same argument applies to DRAM access energy consumption. By reducing the word-line activation frequency, the MB-oriented DRAM storage adopted in this work can effectively speed up the operation and save the energy consumption. As shown in Table 2 4-layer DRAM stacking can achieve much less footprint and slightly better energy efficiency than its 1-layer counterpart. If each memory bank is partitioned into smaller sub-banks, the burst read energy becomes higher, since more energy is consumed on the memory global routing. The access time does not change monotonically with size of sub-banks as observed in the table, because when the sub-banks become smaller, the delay in peripheral circuits tends to increase, even though it takes less time to access the memory core.

As pointed out in the above, the 3D stacked DRAM data storage strategy can naturally support various block matching schemes, including exhaustive full search (FS), three step search (TSS) [9], new three step search (NTSS) [10], four step search (FSS) [11, 12], and diamond search (DS) [13, 14]. Moreover, hybrid block matching can also be seamlessly supported, e.g., we can first run three step search motion estimation on the search region of reference frame and then run full search motion estimation again on the search region with the least SAD value. Since all the search schemes are seamlessly supported, the trade-off between motion estimation accuracy and computational complexity (hence power consumption) can be easily and gracefully configured on the fly. In this work, we use two HDTV 1080p video sequences *Tractor* and *Rush hour* [15] for the purpose of demonstration, where 15 frames are extracted and analyzed in each video sequence. Fig. 6 and Fig. 7 show the peak signal-to-noise ratio (PSNR) of various multi-frame motion estimation algorithms over 3D DRAM energy consumption to process one image frame

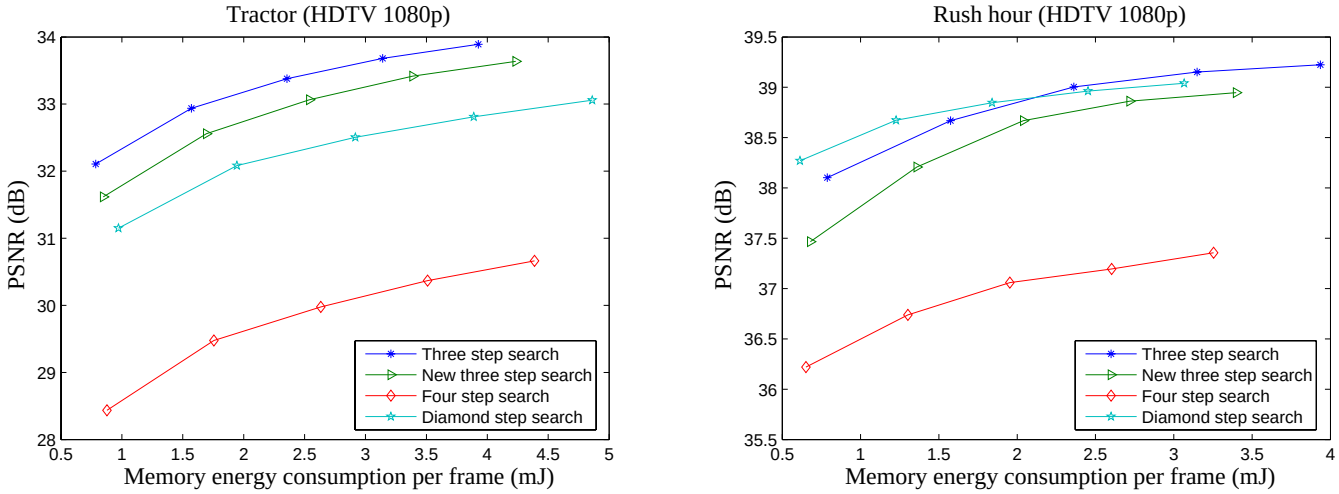


Figure 6: Performance of various advanced motion estimation algorithms vs. 3D DRAM access energy consumption without on-chip SRAM buffer, where we assume there is 1-layer 3D DRAM stacking and 1 sub-bank per bank in 3D DRAM.

without using the on-chip SRAM buffer. Each curve contains five points, corresponding to the scenarios using 1, 2, 3, 4 and 5 reference frames respectively. The memory access energy consumption for each point is obtained based on DRAM parameters with 1-layer DRAM stacking and 1 sub-bank per bank. Due to the regular memory access pattern in full search, explicit memory access can be greatly reduced by data reuse [16] in motion estimation engine. In this study, we set the full search motion estimation design approach can fully exploit such data reuse. However, since the search region is 80×80, the number of full search computation needed to find a best match of a current MB is much larger than the number of other advanced algorithms, and energy consumption of full search is still much higher, so we plot the curves for full search in Fig. 7 and curves for the other algorithms in Fig. 6 to make the figures more legible.

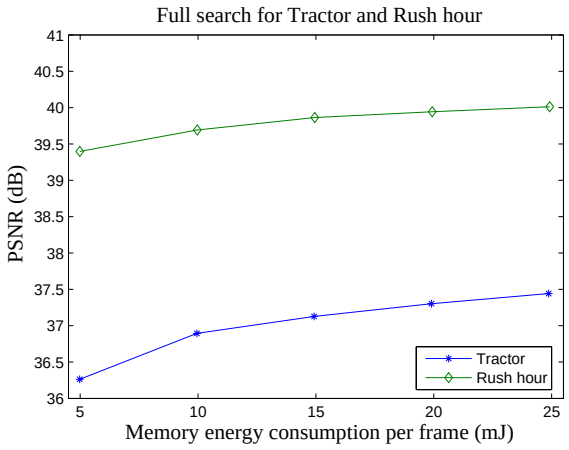


Figure 7: Performance of full search algorithm vs. 3D DRAM access energy consumption without on-chip SRAM buffer, where we assume there is 1-layer 3D DRAM stacking and 1 sub-bank per bank in 3D DRAM.

Based upon the above 3D DRAM parameters, we estimate the overall memory sub-system power consumption and obtain the comparison with conventional design practice. Regardless to whether 3D DRAM stacking is used or not, we assume the image frames are originally stored in an off-chip 4Gb commodity DRAM, which also stores data and instructions for other cores in the envisioned heterogeneous multi-core systems. In the case of conventional design practice, the motion estimation accelerator contains an on-chip SRAM buffer to store the current MB and a 80×80 search region in each reference frame (i.e., a total 250Kb when using 5 reference frames). In the case of 3D-based design, we consider two scenarios: (i) we eliminate the on-chip SRAM buffer in order to reduce the logic die area, where motion estimation computation blocks directly obtain image data from 3D stacked DRAM, and (ii) we still keep the on-chip SRAM buffer in order to reduce the frequency of 3D stacked DRAM access. The estimation and comparison results are list in Table 3 and Table 4, in which full search and three step search algorithms are used respectively. The access power listed in the table is the power consumed for memory read/write under the real-time constraint (i.e., 30 frames per second for HDTV1080p), including address decoder, sense amplifiers, repeaters, routing, etc. The I/O power of DRAM and SRAM is aggregated and listed in the table. It clearly shows the energy efficiency advantages when 3D stacked DRAM is being used, especially in three step search case, mainly because the use of 3D stacked DRAM can largely reduce the frequency of off-chip 4Gb commodity DRAM access and data access to the small 3D stacked DRAM dedicated for motion estimation is much more energy-efficient than access to off-chip 4Gb commodity DRAM. Nevertheless, accessing 3D DRAM directly without on-chip SRAM consumes a large amount of power if full search is used. It is because a large search region imposes more frequent data

Table 3: Comparison between current design practice and the design using 3D stacked DRAM with full search.

		Current practice	Design using 3D stacked DRAM	
			(w/o on-chip SRAM)	(w/ on-chip SRAM)
Off-chip DRAM	Capacity (Gb)	4	4	4
	Access power (mW)	274.56	21.90	21.90
	I/O Power (mW)	124.02	9.89	9.89
3D DRAM	Capacity (Mb)	N/A	100	100
	Access power (mW)	N/A	910.93	18.50
	Leakage power (mW)	N/A	31.03	31.03
	Barrel shifter power (mW)	N/A	???	???
	I/O power (mW)	N/A	146.03	2.96
On-chip SRAM	Capacity (Kb)	250	N/A	250
	Footprint (mm^2)	0.91	N/A	0.91
	Access power (mW)	187.12	N/A	187.12
Total memory access power (mW)		585.70	1119.79	271.39

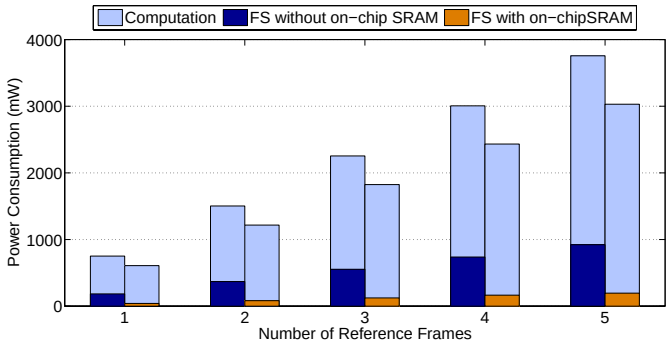


Figure 9: Average power consumption of entire motion estimation using full search algorithm.

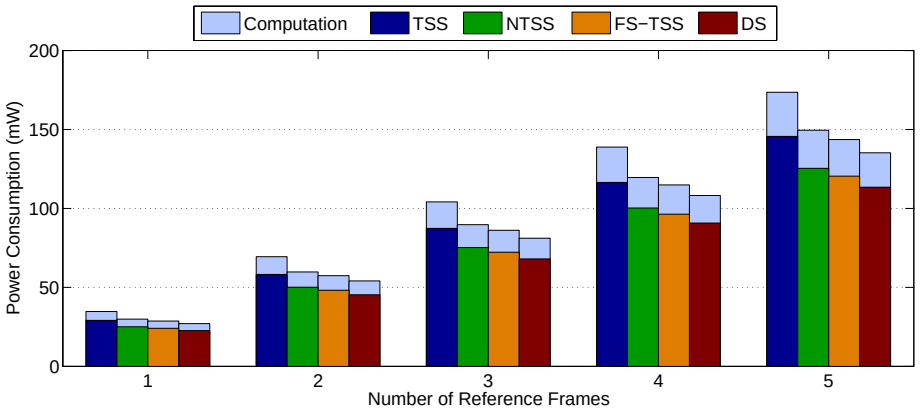
access from 3D DRAM. Moreover, the above two different scenarios (i.e., with and without on-chip SRAM buffer when using 3D stacked DRAM) essentially represent different trade-offs between the logic die area reduction and total memory access power consumption.

Using Synopsys tool sets and a 65nm CMOS standard cell library, we synthesize parallel motion estimation computation engines and estimate the power consumption. Combining the above memory sub-system power consumption results, we estimate the overall average motion estimation power consumption as shown in Fig. 8 and Fig. 9, where we consider both with and without on-chip SRAM buffer. Because the search steps of NTSS, FS-TSS and DS algorithms may vary during the run time, we show the average values obtained from computer simulations. The lower and upper parts of each bar in the figures represent memory access and computation logic power consumption respectively. The power consumption is obtained at the clock frequencies that are just fast enough for real-time processing in all the studies.

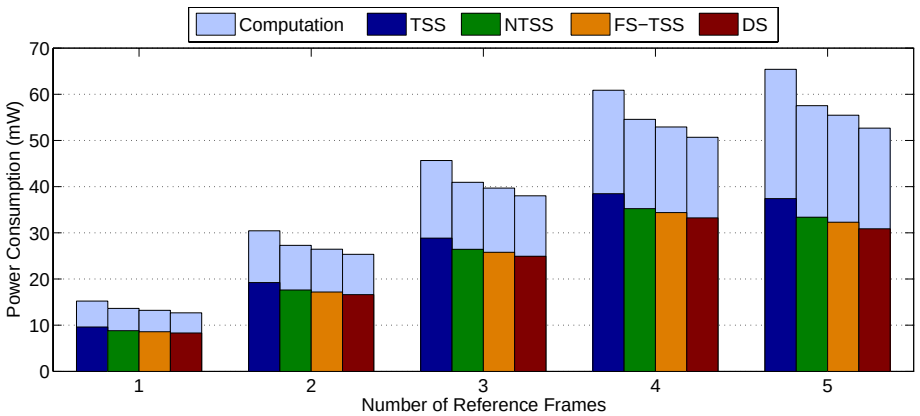
Finally, we evaluate the potential of leveraging the bit-plane DRAM storage structure to enable graceful performance vs. energy consumption. We only consider the scenario of 3D stacked DRAM without on-chip SRAM buffer. We carry out simulations with full search, three step search and new three step search algorithms with one reference frame, and we take the DRAM parameters of 1-layer stacking and 1 sub-bank per bank listed in Table 2. For full search, we simply study the use of 4 bits, 5 bits,$\cdots$, 8 bits of every pixel in motion estimation. For three step search, because of the relatively large search region (i.e., 80×80), every MB actually takes five steps to find the best matched MB. In the first step, since 9 candidate MBs are well separated away from each other in a reference frame and their SADs between current MB tend to largely differ, less number of bits may be sufficient, hence 4 bits of each pixel are used in the first step. On the other hand, search in the last two steps is limited into a small region and the accuracy directly affects the motion estimation performance, therefore we used full precision in the last two steps. We only vary the number of bits used for motion estimation in the second and third steps. For new three step search, which evolves from three step search, the number of steps can be either 2 or 5 for every current MB. So we only vary the precision in the first step, and we keep the full precision during the following 1 or 4 step(s). The PSNR vs. memory access energy consumption results are shown in Fig. 10. The five points on each full-search curve correspond to using 4, 5, 6, 7 and 8 bits in motion estimation; the five points on each new-three-step-search curve correspond to using 4, 5, 6, 7 and 8 bits in the first step; the left five points on each three-step-search curve correspond to using 4, 5, 6, 7 and 8 bits in the second and third steps and the rightmost point represents the result when the full precision is being used in all the five steps. The simulation results clearly show the trade-offs in various design scenarios. In particular, for three step search, if we only reduce the first step precision from 8 bits to 4 bits and keep the full precision in the rest steps, PSNR performance does not degrade at all, while it can save almost 15% of memory access power consumption as well as computation logic power consumption.

Table 4: Comparison between current design practice and the design using 3D stacked DRAM with three step search.

		Current practice	Design using 3D stacked DRAM	
			(w/o on-chip SRAM)	(w/ on-chip SRAM)
Off-chip DRAM	Capacity (Gb)	4	4	4
	Access power (mW)	274.56	21.90	21.90
	I/O power (mW)	124.02	9.89	9.89
3D DRAM	Capacity (Mb)	N/A	100	100
	Access power (mW)	N/A	146.49	18.50
	Leakage power(mW)	N/A	31.03	31.03
	Barrel shifter power (mW)	N/A	???	???
	I/O power (mW)	N/A	23.48	2.96
On-chip SRAM	Capacity (Kb)	250	N/A	250
	Footprint (mm^2)	0.91	N/A	0.91
	Access power (mW)	29.97	N/A	29.97
Total memory access power (mW)		428.55	232.79	114.24



(a)



(b)

Figure 8: Average power consumption of entire motion estimation using advanced algorithms (a) without on-chip SRAM buffer, and (b) with on-chip SRAM buffer.

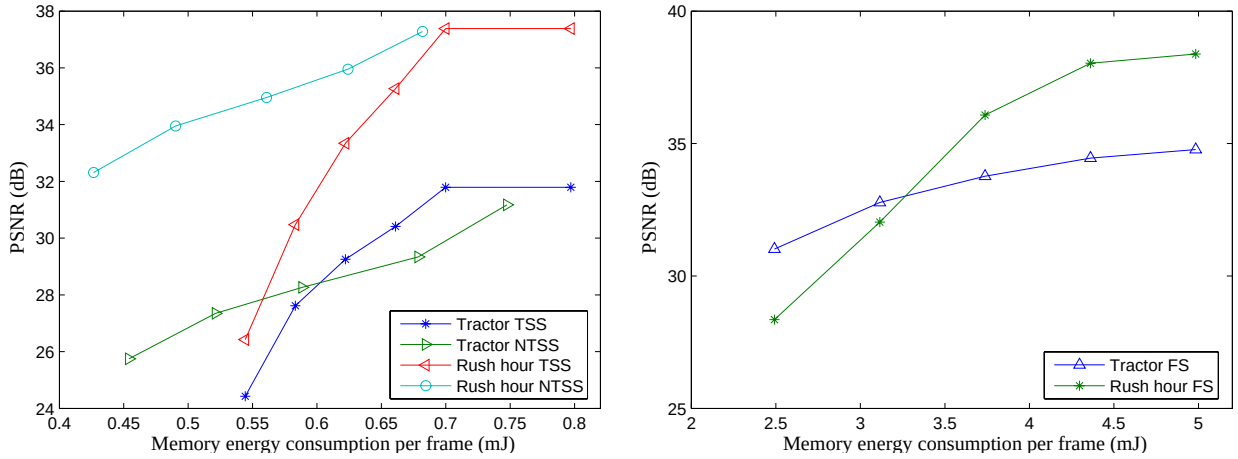


Figure 10: Motion estimation PSNR vs. memory access energy when varying pixel luminance intensity precisions, where different points on the same curve correspond to different bit precisions for block matching: (a) Three step search and new three step search for both video sequences; (b) Full search for both video sequences.

4. Conclusion

With 3D memory stacking, memory access may no longer be a bottleneck for video encoding, in particular for motion estimation. This provides new opportunities to explore efficient motion estimation accelerator architecture design. In this paper, we investigate motion estimation accelerator architectures to effectively utilize the 3D integrated DRAM. We develop one specific 3D DRAM data storage organization and frame storage strategy geared to motion estimation, which can be seamlessly coupled with parallel motion estimation computation engines. The presented design strategy has a good energy efficiency and can seamlessly support various motion estimation algorithms with variable block size. Using DRAM performance modeling/estimation tool and ASIC design at 65nm technology node, we demonstrate the effectiveness of such 3D integrated motion estimation accelerator design for multi-frame motion estimation.

5. Acknowledgement

The authors sincerely appreciate the anonymous reviewers for their very insightful and constructive comments that greatly helped to improve the quality and presentation of this work.

References

[1] Y. H. et.al, A 1.3TOPS H.264/AVC single-chip encoder for HDTV applications, in: Proc. of IEEE Intl. Solid-State Circuit Conf. (ISSCC), San Francisco, 2005, pp. 128–129.

[2] J.-Q. Lu, 3-D Hyperintegration and Packaging Technologies for Micro-Nano Systems, Proceedings of the IEEE 97 (2009) 18–30.

[3] M.Facchini, T.Carlson, A.Vignon, M.Palkovic, F.Catthoor, W.Dehaene, L.Benini, P.Marchal, System-level power/performance evaluation of 3d stacked drams for mobile applications, in: Proc. of Design, Automation & Test in Europe Conference & Exhibition, 2009, pp. 923–928.

[4] S. Gu, P. Marchal, M. Facchini, F. Wang, M. Suh, D. Lisk, M. Nowak, Stackable memory of 3D chip integration for mobile applications, in: Proc. of IEEE International Electron Devices Meeting (IEDM), 2008, pp. 1–4.

[5] E. Jaspers, P. de With, Bandwidth reduction for video processing in consumer systems, IEEE Trans. on Consumer Electronics 47 (4) (2001) 885–894.

[6] Z.-L. He, C.-Y. Tsui, K.-K. Chan, M. Liou, Low-power vlsi design for motion estimation using adaptive pixel truncation, IEEE Trans. on Circuits and Systems for Video Technology 10 (5) (2000) 669–678.

[7] R. Anigundi, H. Sun, J. Lu, K. Rose, T. Zhang, Architecture Design Exploration of Three-Dimensional (3D) Integrated DRAM, in: Proc. of IEEE International Symposium on Quality Electronic Design (ISQED), 2009.

[8] CACTI: An integrated cache and memory access time, cycle time, area, leakage, and dynamic power model, http://www.hpl.hp.com/research/cacti/.

[9] L.-P. Chau, X. Jing, Efficient three-step search algorithm for block motion estimation video coding, in: Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003, pp. 421–424.

[10] R. Li, B. Zeng, M. L. Liou, A new three-step search algorithm for block motion estimation, IEEE Trans. on Circuits and Systems for Video Technology 4 (1994) 438–442.

[11] L.-M. Po, W.-C. Ma, A novel four-step search algorithm for fast block motion estimation, IEEE Trans. on Circuits and Systems for Video Technology 6 (1996) 313–317.

[12] K.-T. Wang, O.-C. Chen, Motion estimation using an efficient four-step search method, in: Proc. of IEEE International Symposium on Circuits and Systems, 1998, pp. 217–220.

[13] J. Y. Tham, S. Ranganath, M. Ranganath, A. A. Kassim, A novel unrestricted center-biased diamond search algorithms for block motion estimation, IEEE Trans. on Circuits and Systems for Video Technology 8 (1998) 369–377.

[14] J. Y. Tham, S. Ranganath, M. Ranganath, A. A. Kassim, A new diamond search algorithm for fast block-matching motion estimation, IEEE Trans. on Image processing 9 (2000) 287–290.

[15] Sveriges Television (SVT), http://www.svt.se.

[16] J. Tuan, T. Chang, C. Jen, On the data reuse and memory bandwidth analysis for full-searchblock-matching VLSI architecture, IEEE Trans. on Circuits and Systems on Video Technology 12 (1) (2002) 561–72.