A 130.3 mW 16-Core Mobile GPU With Power-Aware Pixel Approximation Techniques

Yu-Jung Chen, Chao-Hsien Hsu, Chung-Yao Hung, Chia-Ming Chang, Shan-Yi Chuang, Liang-Gee Chen, *Fellow, IEEE*, and Shao-Yi Chien, *Member, IEEE*

***Abstract*—Intensive pixel shading dominates the power dissipation of the graphics pipeline as the screen resolution grows. In this work, we propose a 130.3 mW 16-core mobile GPU with three pixel approximation techniques and a corresponding tile-based rasterization architecture. The proposed architecture can trade-off between power consumption and visual quality to provide power-aware capability, and is fabricated with TSMC 45 nm technology. The feasibility and effectiveness of these techniques are verified in this chip prototype. The implementation results show that, with satisfactory visual quality, 52.32% of the power consumption of the shader processors can be empirically reduced with an experimental *Approximated Precision Shader architecture* and a *Screen-space Approximated Lighting* technique. Furthermore, the *Approximated Texturing* technique can reduce 24.57% of L1 cache updates in our evaluation.**

***Index Terms*—Approximated Precision Shader, Approximated Texturing, Screen-Space Approximated Lighting.**

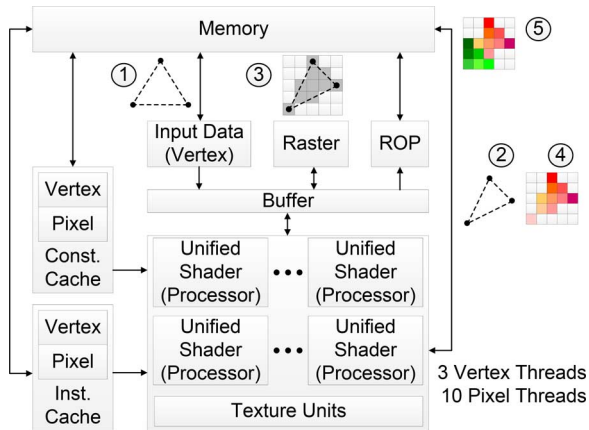

Fig. 1. A general GPU architecture and hardware pipeline.

I. Introduction

THE MOBILE graphics processing unit (GPU) has become an essential component in mobile devices especially as the display resolution rapidly grows [1]. A typical GPU pipeline is shown in Fig. 1 with steps indicated by the numbers. The thread number are arbitrary chosen for explaining the hardware pipeline. 1) First, vertex data, such as vertex positions, normals and texture coordinates are fetched into the input data buffer. For a triangle, three vertex threads are launched among the unified shaders for vertex transformation. 2) The transformed vertices in the internal buffer are assembled into a triangle. 3) With the transformed triangle, the information of the pixels in screen space is generated through interpolation in the raster unit. In this example, ten pixels are generated. Subsequently, ten pixel threads are lunched in the unified shaders for executing pixel programs, which include lighting and texturing effects, to shade pixels. 4) The shaded pixels are stored to internal buffer. 5) According to the corresponding pixel depth, stencil and alpha, ROP unit conducts the visibility test and blends pixel color of each pixel, which is then output to the framebuffer.

There are various researches [2]–[8] aiming at optimizing the mobile GPU design in architecture and circuit levels. Tsao *et al.* [2], [3] target at efficient instructions reordering, accelerating instructions for multimedia applications and efficient configurable memory buffer. Nam *et al.* [4] focus on the micro-architecture design of the shader processor, including instruction set architecture and logarithmic datapath. Chang *et al.* [5], [6] propose a configurable filtering unit, efficient buffer transaction technique and pixel duplication scheme for energy saving. Kim *et al.* [7], [8] enhance memory bandwidth through process technology and extend the vector processor configuration for Augmented Reality (AR) applications.

The energy saving scheme with pixel duplication proposed by Chang *et al.* [5], [6] exploits the trade-off between approximated image quality and power consumption to achieve power efficiency. However, the pixel duplication scheme simply saves power consumption without considering visual quality. Motivated by this idea, we developed a series of pixel approximation

Manuscript received November 16, 2014; revised January 06, 2015; accepted April 08, 2015. Date of publication May 08, 2015; date of current version August 27, 2015. This paper was approved by Associate Editor Stefan Rusu. This work was supported by the Ministry of Science and Technology under Grants MOST 100-2221-E-002-090-MY3 and MOST 103-2221-E-002-268-MY3.

Y.-J. Chen, C.-H. Hsu, and S.-Y. Chien are with the Media IC and System Lab, Graduate Institute of Electronics Engineering and Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan (e-mail: stevechen@media.ee.ntu.edu.tw; chsiensu@media.ee.ntu.edu.tw; sychien@ntu.edu.tw).

C.-Y. Hung was with the Graduate Institute of Electronics Engineering and Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, and is now with MediaTek, Hsinchu, Taiwan (e-mail: cyhung@media.ee.ntu.edu.tw;).

C.-M. Chang was with the Graduate Institute of Electronics Engineering and Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, and is now with ARM Inc., Taipei, Taiwan (e-mail: solo.chang@gmail.com).

S.-Y. Chuang and L.-G. Chen are with the DSP/IC Design Lab, Graduate Institute of Electronics Engineering and Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan (e-mail: kasen.chuang@video.ee.ntu.edu.tw; lgchen@ntu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSSC.2015.2423972

techniques on screen pixels. The concept of pixel approximation is quite similar to the prediction and reconstruction process in image compression, and we can approximate pixel values under certain observations:

- The precision loss of the arithmetic operations in a graphics pipeline is barely perceived in some cases.
- Smooth textures change slightly between different scales.
- Illumination often varies smoothly in spatial domain.

Based on these concepts, a power-aware 16-core GPU is developed with our proposed approximation techniques. Experimental results empirically reveal that these proposed techniques can better approximate certain visual effects, meanwhile lowering the power consumption or memory dynamics. The proposed approximation techniques are featured in this chip prototype as follows

(a) *Approximated Precision Shader (APS)* architecture is an experimental heterogeneous multi-core architecture, where lower power consumption can be achieved by partially distributing workload to lightweight shader cores.

(b) *Approximated Texturing (AT)* mechanism reduces the memory request of texture access by adaptively approximating the texture information with the concept of wavelet transform.

(c) *Scree-space Approximated Lighting (SSAL)* technique can approximate the illumination with adaptive sampling patterns and plane fitting.

Similar to lossy image compression, where the perceptual redundancy of the data is removed, in our work, the redundancy in computation is saved to reduce the power consumption. Compared with the pixel duplication scheme [5], [6], the proposed SSAL further improves the quality loss caused by direct pixel duplication with better sampling and reconstruction strategies, such as linear interpolation, averaging and splatting based on certain sampling patterns. The proposed mobile GPU can achieve low power consumption while visual quality is well preserved with these approximation techniques.

This paper is organized as follows. Section II elaborates the proposed architecture. The proposed tile-based raster unit is first introduced in Section III. Then the three proposed approximation techniques are thoroughly described in Sections IV–VI. Section VII summarize the implementation results and comparisons between state-of-the-art designs. Finally, Section IX concludes this work.

II. Architecture Overview

The architecture overview of the proposed mobile GPU is shown in Fig. 2. The GPU architecture majorly consists of four shader clusters. There are four unified shader cores, which is denoted as *US* in Fig. 2, a texture unit, and a corresponding texture L1 cache, in each shader cluster. Thus, sixteen active vector threads can concurrently perform either vertex program or pixel program in parallel. Our unified shader architetcure is a four-channel Single-Instruction-Multiple-Data (SIMD) processor. The SIMD processor is illustrated in Fig. 3. It has four pipeline stages, including Fetch, Decode, Execute, and Write Back stages. During Fetch stage, the instruction fetch unit incrementally fetches instructions from the instruction cache according to the current program counter. In addition, the fetch

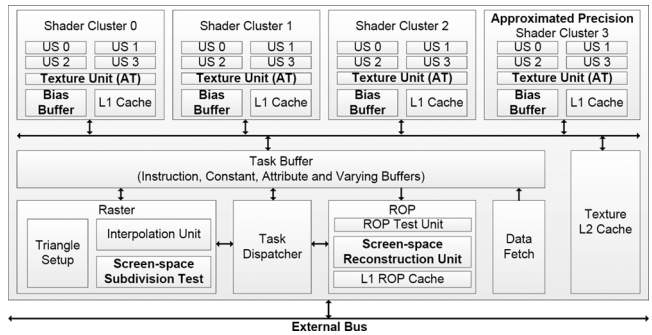

Fig. 2. Architecture overview of the proposed 16-core mobile GPU.

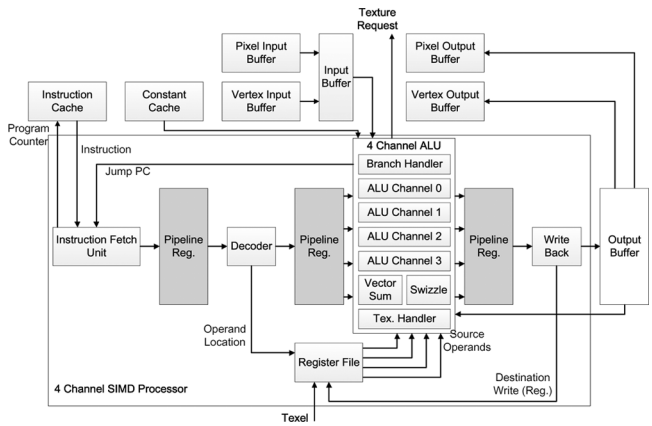

Fig. 3. The four-channel SIMD architecture.

unit needs to be aware of the occurrence of *Jump* instruction. The instruction is then decoded during the Decode stage to retrieve the op (operation) code, source and destination operands. With the source operands, the Execution stage activates the corresponding datapath or the ALU channels based on the vector type (i.e., vec2, vec3 or vec4). The operands can be accessed from the register file, constant cache (i.e., *uniform* in GLSL), input buffer (i.e., vertex *attribute* or pixel *varying* in GLSL) or the output buffer. These operands are in 32 bit representation. Each ALU channel has fundamental arithmetic units such as floating point multiplier, adder, comparators, logic operations and absolute values, as shown in Fig. 4. In addition to the ALU channels, the execution stage can perform vector swizzle and summation operations; texture handler is responsible for issuing texture requests to the texture unit with texture ID (i.e., texture *sampler* in GLSL), coordinate and Level of Detail (LOD). Finally, the Write Back stage forwards the result from Execution stage to the register file or the output buffer. The output buffer then dispatches the computed output to the vertex or pixel output buffer depending on the current thread type (i.e., vertex or pixel thread).

The proposed *Approximated Precision Shader (APS)* technique is realized in *Shader Cluster 3*. Current configuration is adopted for empirically evaluating the effectiveness of APS. Besides, for each texture unit, the proposed *Approximated Texturing (AT)* mechanism is implemented. In addition to the texture L1 cache, an extra bias buffer is designed for supporting the *Approximated Texturing* technique. To reduce the external texture bandwidth, a texture L2 cache is integrated as well. Both

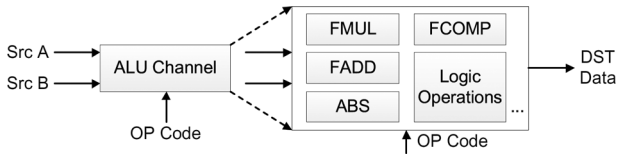

Fig. 4. ALU channel of the SIMD processor.

vertex attribute and pixel varying buffers are based on our previous proposed configurable memory array architecture [2]. The data fetch unit is responsible for accessing data from the external memory. The threads among shader cores are scheduled by the task dispatcher. To achieve the proposed *Scree-space Approximated Lighting (SSAL)* technique, the dispatcher is utilized to orchestrate the unified shaders, the screen-space subdivision test unit and the screen-space reconstruction unit.

III. Tile-Based Raster Unit

The raster unit is used to generate the variables, which is defined as *varying* in the GLSL standard, of pixels covered by the projected triangle in screen space. The rasterization process has been comprehensively studied [9]–[11]. The concept is to examine the potential screen pixel positions of a triangle based on the edge equations. Tile-based rasterization [12], [13], where pixels are scanned tile-by-tile, is proved to provide better cache efficiency with higher data locality [14]. The tile-based scheme can be taken as a two-level scanning process: tile traversal and interior traversal passes, where the tile traversal pass scans the screen tiles in a designed order, and the interior traversal pass scans the pixels within the tile. All these previous works [9]–[13] concentrate on discussing the rasterization algorithm. In this work, a tile-based hardware raster with two-level scanning process is realized with a set of ALU. Our raster unit traverses and interpolates the tiled pixels with a designed scanning order from the primitive edges, with which we can regularly schedule the ALU set to increase or decrease the value of pixel varyings according to the corresponding plane coefficients. Moreover, the interior traversal unit is modified and coupled with the thread dispatcher for SSAL to detect the approximation patterns.

A. Rasterization With Edge and Plane Equations

Generally, the edge equations are adopted for the raterization process and can be generalized as

$$e(x, y) = \alpha x + \beta y + \gamma = 0 \tag{1}$$

where the parameters α, β, and γ are derived from the three edges of the input triangle during the triangle setup stage. For each projected pixel (x, y) on the edge, it maintains the property that $e(x, y) = 0$. Therefore, the projected pixels on the positive side of the normal return $e(x, y) > 0$, and $e(x, y) < 0$ for the pixels on the negative side. A pixel is determined inside the projected triangle if it is on the positive side of the three edges.

In our design, a general plane equation form is utilized to to interpolate the projected pixels. As shown in Fig. 5, assuming that an input triangle with vertices p^n and the corresponding scalar variables s^n, where $n = 0 - 2$), the implicit form of a plane equation for a triangle is $ax + by + cs + d = 0$, which can

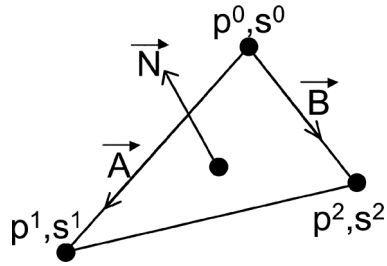

Fig. 5. A triangle with scalar variables s^0, s^1, and s^2 at vertices p^0, p^1, and p^2, respectively.

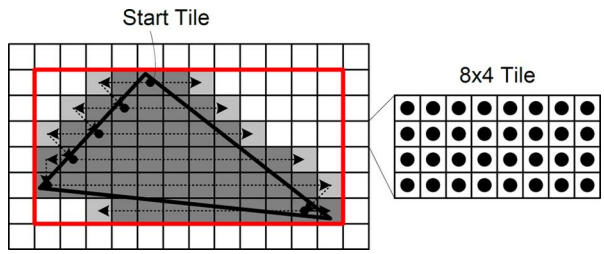

Fig. 6. Proposed tile scan algorithm.

be derived with $\vec{N} \cdot \vec{P}$, where $\vec{N} = \vec{A} \times \vec{B}$. The scalar variable of a pixel (x, y) can thus be interpolated based on the equation, $s(x, y) = -a/c\, x + (-b/c)y - d/c = Mx + Ny + C$.

Our raster engine adopts the two-level scheme. During the tile traversal pass, the screen tiles covered by a triangle are identified. As shown in Fig. 6, we first derive the bounding box of a triangle in tile level, as the red box indicates. Next, the scanning process starts from the tile covering the top vertex, which is assigned as the Start Tile, and then processes by tile lines. The scan order is from the Start Tile to the right, and then from the Start Tile to the left. The scanned tile is examined by verifying the four corner pixels of a tile with the edge equations to determine if the tile is completely inside the triangle, and the scanning process is terminated while the scanned tile is outside the triangle. The last tile covered by the triangle in the current tile row is recorded, and the tile below it is assigned as the Start Tile of the next tile traversal process, as shown in Fig. 6.

Regarding the interior traversal pass, the pixels within a tile which passes the tile traversal stage are then scanned. To reduce the computational complexity, the value of each pixel sample can be derived merely with an addition because the value of the neighboring pixel is already evaluated. When a pixel (x, y) is previously scanned, the value of the next pixel $(x+1, y)$ can be derived with the following equation:

$$s(x + 1, y) = M(x + 1) + Ny + C = s(x, y) + M. \tag{2}$$

The values of all pixels within a tile can be derived through iterating this process.

B. Proposed Hardware Architecture of Tile-Based Raster Unit

The architecture of our tile-based raster unit is shown in Fig. 7. To meet the cost efficiency requirement on a mobile platform, it is designed with a folded and pipelined architecture. Our design is mainly composed of four modules, including triangle setup, tile traversal, interior traversal and interpolation

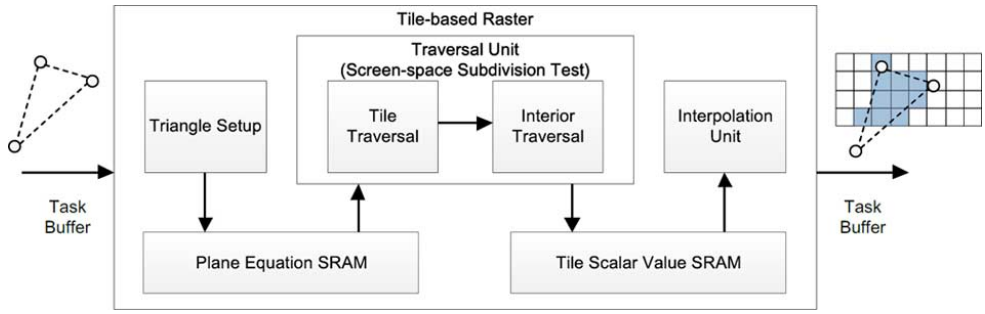

Fig. 7. Hardware pipeline of the proposed raster unit.

unit. Two SRAM modules are introduced among these modules as the intermediate coefficient storage.

To setup all plane equations, including the scalar values and the edge functions as described in Section III-A, the triangle setup module is designed, which is composed of the cross product, addition and division arithmetic units. With properly scheduling and folding the computation on the ALU set, the coefficients of the plane equations are derived and stored in the intermediate plane equation SRAM.

The traversal unit is composed of two modules, including tile traversal and interior traversal. The tile traversal module realizes the tile scanning process by checking the plane equations of the four corner pixels of each tile. If a tile potentially has pixels covered by the triangle, it is further forwarded to the interior traversal module to examine the covered pixels. The covered pixels are first recorded as a binary valid map for each tile (i.e., 32 bits for a 4×8 tile). The order of interior traversal starts from the leftmost column to the right. The interior traversal module generates the corresponding scalar values of the plane equation for the four pixels of the leftmost column and stores the values into the tile scalar value SRAM. With the values of the stored leftmost pixels, the interpolation unit can directly realize (2). Pixel variables, such as position and the associated *varying*, are generated for a 16 pixel set in parallel.

IV. Approximated Precision Shader

In this work, we target at OpenGL ES 2.0 specification and employ 32 bit floating point representation for the computation. Although the variables of shader programs are represented as floating point numbers, the shaded pixel color are converted into a limited integer representation ranging from 0–255 for display. Since the real-time rendering applications are not accuracy-oriented, Pool *et al.* [15], [16] propose a precision reduction technique in either vertex or pixel shader to reduce power consumption. On the other hand, ARM proposes a heterogeneous processor architecture to enhance energy efficiency [17]. Heterogeneous architecture becomes a future trend for improving energy efficiency.

There are various approximation domains for a shader architecture. In the early OpenGL specification, half-precision floating-point format for computation is adopted; general fixed-point concept can also be applied. The shader processor design [4] proposed by Nam *et al.* approximates the arithmetic operations in logarithmic domain. The design purpose is to approximate computation while executing a program, from the computation perspective. From the perspective of rendering, since the PSNR quality of the output pixels is less sensitive to precision loss, Pool *et al.* [16] propose an adaptive precision selection approach, including a corresponding compiler design. They simulate the adaptive precision technique on the Attila GPU simulator [18]. Although the precision reduction strategy is similar to the early half-precision floating-point format, the purpose differs. The precision reduction method approximates the floating-point format to reduce power consumption, where the half-precision floating point is a previous OpenGL target specification.

Motivated by these design concepts, the proposed *Approximated Precision Shader (APS)* architecture for pixel program is realized in *Shader Cluster 3*. With this technique, the power consumption of the shader processor array is reduced with tolerable quality degradation. As Fig. 8(a) illustrates, the 32 bit IEEE 754 representation is composed of 1 sign bit, 8 exponent bits and 23 mantissa bits. If we truncate the 16 least significant mantissa bits, we can have an approximated floating point representation. Take π as an example, the full 32 bits representation is 3.1415927, and it becomes 3.140625 with our truncated 16 bits representation. Regarding the pixel color, the right-hand side of Fig. 8(a) shows an example for comparison between the full bits representation and truncated version. The color difference is barely perceived. Note that the logarithmic arithmetic based SIMD processor [4] proposed by Nam *et al.* is an optimized processor architecture, where the approximation loss is merely induced due to the logarithmic arithmetic conversion. In our design, APS discards mantissa bits, which intends to reduce the size of registers and combinational logics for saving power. The main concept of APS is to approximate the output from pixel shader. Since the floating point pixel value is further quantized to 0 to 255 during ROP, the approximated precision loss has less impact on the quality degradation of pixels.

The proposed architecture is shown in Fig. 8(b). The logical pixel threads are distributed to the shader clusters by task dispatcher with a task FIFO of the rasterized pixels. For the APS processor, the vector ALUs and the corresponding registers are resized as the truncated 16 bit precision. We preserve the precision of registers for pixel positions and interpolated texture coordinates. Since the rasterized position determines the pixel position on framebuffer, it is sensitive to precision loss, and a MOV (move) instruction is first performed in the pixel program to store the full precision position, including depth, into output buffer. Besides pixel position, texture coordinate, which

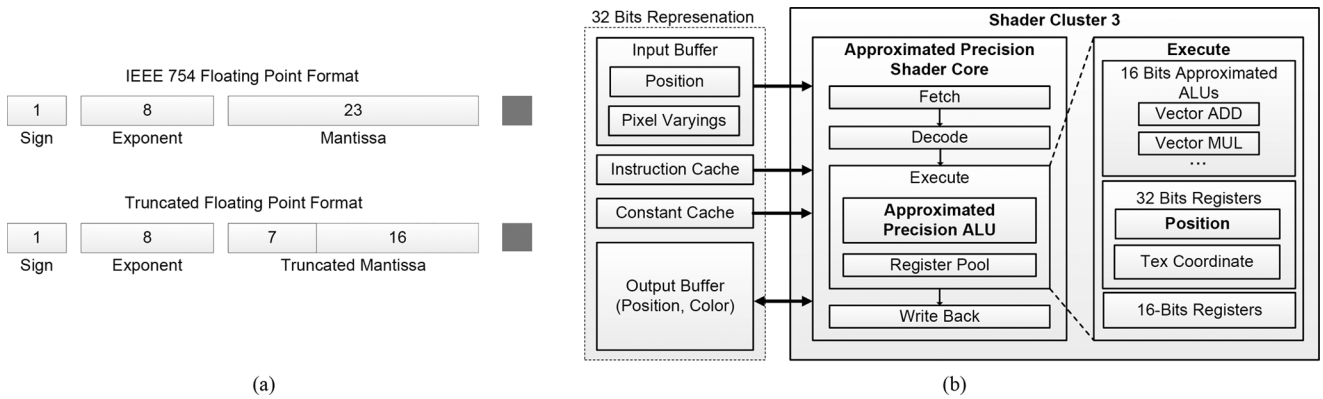

Fig. 8. The proposed Approximated Precision Shader technique. (a) Simple illustration of truncated floating point representation. (b)The proposed APS architecture.

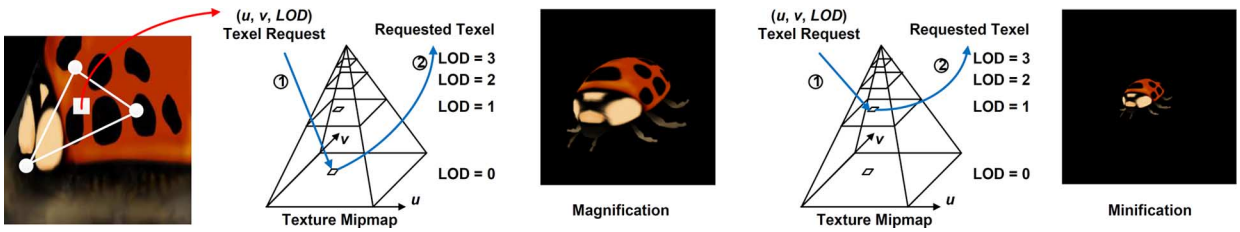

Fig. 9. Illustration of the standard texture mipmapping concept. Magnification accesses lower level of texture image, while minification accesses higher level of texture image.

locates the texels on texture images, is sensitive to precision loss as well. Another set of full-precision register is also preserved for texture coordinates to issue texture fetch. Before storing the computed color value to the output buffer, the approximated 16 bit color value needs to be padded with zeros to a 32 bit representation. Our implementation results of the approximated precision shader reveal that the area is 61% compared with a full precision shader. The average power consumption of performing a pixel program in our SIMD and APS SIMD processor are 2.57 mW and 1.42 mW, respectively. Thus, the power consumption of the APS SIMD processor is about 55% compared with our full precision SIMD processor. In this chip prototype, our purpose is to reduce power consumption through approximating pixel values. The APS concept is different from the heterogeneous cores of the ARM processor, which is designed for various workload characteristics. Although our shader processors perform the same pixel program, the APS cores deliver the approximated pixel output with lower power consumption. Therefore, the APS can be a simple scheme with less implementation overhead on the shader processor for a low power design.

The proposed APS technique can be regarded as a static hardware implementation of the reduced precision method proposed by Pool *et al.* [16]. In their simulation on the Attila GPU architecture [18], with a similar but different test case of performing screen-space ambient occlusion (SSAO), which is also a smooth illumination effect on a single object, it can achieve power saving in shader processor of 49% and 71% based on their ALU power model with reduced precision, respectively 21 and 13 mantissa bits. Therefore, in their approach for the SSAO case, the average precision using for computation are 29 and 21 bits. Based on our SIMD architecture, we reduce the power consumption around 45% with the truncated 16 bit APS shader. In terms of power saving, the comparison is within an acceptable range.

Note that, although the precision of the rasterized position and depth are preserved, complex shader computation might cause error propagation from the precision loss. Currently, our compiler is not an optimized design to relieve this issue, which still needs to be handled by the programmers. In the future, dedicated compiler design will be further considered.

V. Approximated Texturing

Mipmapping [19] is a texture filtering process to generate correct texture scale for each pixel with a pre-filtered texture pyramid. The driver constructs a texture pyramid for each LOD (Level of Detail) before the rendering pipeline, as shown in Fig. 9. The LOD parameter is used to indicate the scale within the texture pyramid, and to fetch texels for the trilinear filtering process. Each trilinear filtering operation requires four texels from two neighboring levels, and total eight texels are required. With the rasterized texture coordinate of a pixel—(u, v, LOD), each request from the shaders for the texture unit leads to at most eight texel lookups.

The proposed *Approximated Texturing (AT)* technique targets at reducing the cache updating activities with satisfactory approximated visual quality through accessing smaller memory footprint of the texture image, which means accessing the texture contents in higher LOD. First, the texture content is pre-analyzed with the wavelet transform to evaluate the texture complexity. We denote the original texture image as I^0 and the

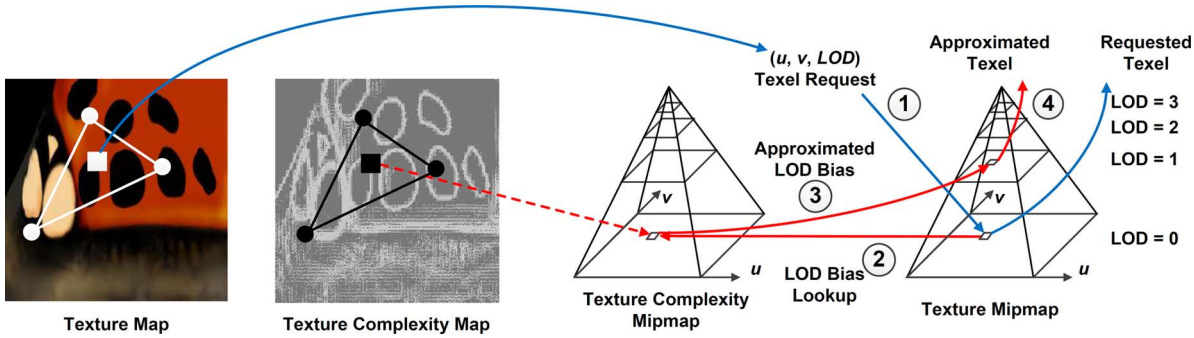

Fig. 10. Concept of the proposed indirectly approximated texturing technique.

mipmaps as $\mathbf{I} = \{I^0, I^1, \ldots\}$ where I^{k+1} is in half resolution of I^k. Given the texture mipmaps, the high-pass filtered images of I^k can be constructed, namely

$$I_x^{k+1}(x,y) = \frac{1}{2}(I^k(2x+1,2y) + I^k(2x+1,2y+1)) - \frac{1}{2}(I^k(2x,2y) + I^k(2x,2y+1)) \quad (3)$$

$$I_y^{k+1}(x,y) = \frac{1}{2}(I^k(2x,2y+1) + I^k(2x+1,2y+1)) - \frac{1}{2}(I^k(2x,2y) + I^k(2x+1,2y)) \quad (4)$$

$$I_{xy}^{k+1}(x,y) = \frac{1}{2}(I^k(2x,2y) + I^k(2x+1,2y+1)) - \frac{1}{2}(I^k(2x+1,2y) + I^k(2x,2y+1)) \quad (5)$$

where $I^k(x,y)$ denotes the texel value at LOD k and coordinate (x,y). These high-passed images describe the local details. The set of images $\{I_x^k, I_y^k, I_{xy}^k | k \in 1,2,3,\ldots\}$ denotes the wavelet transform images in different sub-bands of texture I^0. The total energy of the high-pass images $I_x^{k+1}(x,y)$, $I_y^{k+1}(x,y)$ and $I_{xy}^{k+1}(x,y)$ is then quantized to the levels of texture complexity. The dimension of the texture complexity map is the same as the mipmapped texture pyramid. Each pixel on the texture complexity pyramid represents the corresponding bias level to bias the LOD. With the corresponding complexity level for each texel, we can replace the lower LOD texel requests to higher LOD to get similar quality with smaller memory footprint. In this work, we use 2 bits to represent the LOD bias, which means each texel can be approximated with texels at 3-level upper at most.

Fig. 10 illustrates an example of a texture with the corresponding texture complexity map. It shows that the edges and textured regions are highlighted, indicating that these parts should not be approximated aggressively with higher LOD texture. On the other hand, the smooth texture regions are darker in the texture complexity map, which means that these regions can be better approximated. The numbers indicate the approximation process, and there is an extra indirect texture access (i.e., Step 2) to the texture complexity mipmap to lookup the LOD bias. In our implementation, the 2 bit bias represents the LOD bias levels ranging from 0 (original texture fetch) to 3. The texture buffer bandwidth can then be reduced as the texel requests shift to higher LOD, and the cache performance is improved by accessing smaller memory footprint. As the

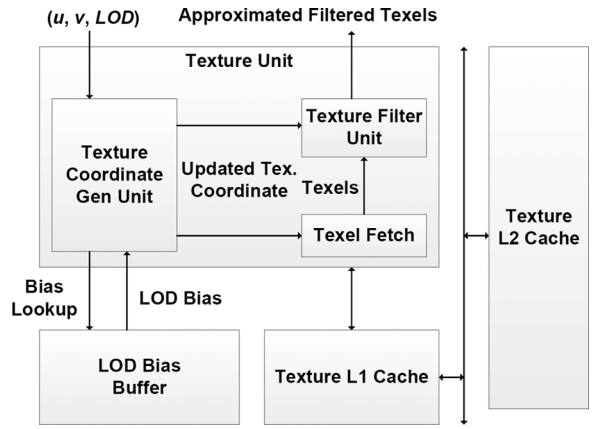

Fig. 11. The proposed architecture of indirectly approximated texturing technique.

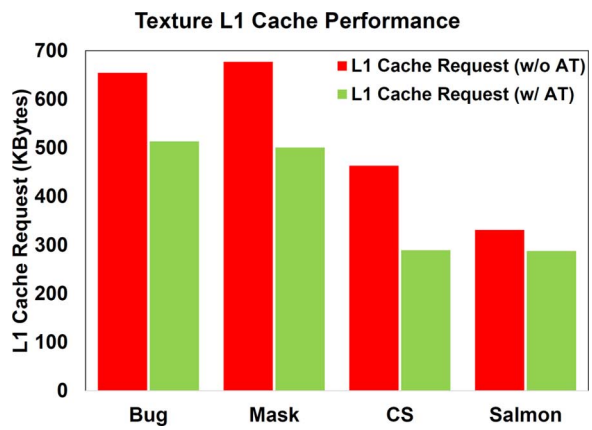

Fig. 12. Average texture L1 cache request in KBytes.

example illustrates, the LOD is shifted from 0 to 1 based on the bias. The quality degradation is well controlled with considering the characteristics of texture content recorded in the texture complexity map.

Fig. 11 illustrates the proposed architecture. The LOD bias offset in the *LOD Bias Buffer* is first retrieved back for the texture coordinate generation unit. Regarding the 2 bit LOD bias, it is quite compact to be buffered compared with the 24 bit or 32 bit texture (i.e., RGB or RGBA format). The texture coordinate generation unit can then compute the texture coordinate and LOD based on the LOD bias. Later, the texel fetch unit issues texel requests from the L1 cache with the computed texture

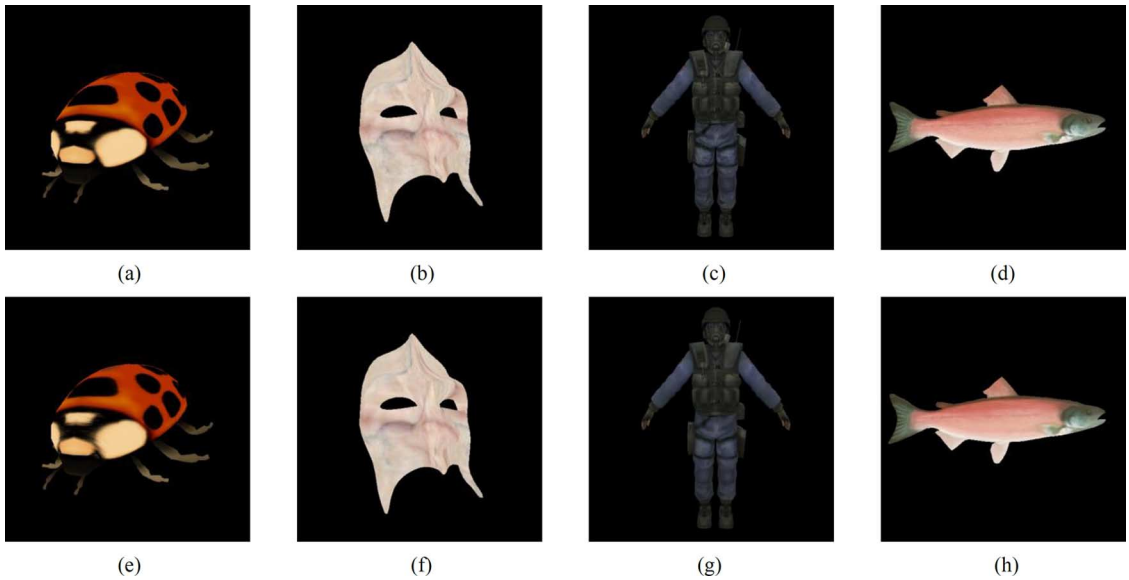

Fig. 13. Quality comparisons between references and approximated texturing technique. From left to right, the triangle numbers of these models are 0.8 K, 2 K, 2.7 K and 1.5 K. (a) to (d) are the reference texture mapped results, and (e) to (h) are the corresponding approximated textured results. The PSNR, which includes the quality degradation caused by the APS technique, for (e) to (h) are 29.61 dB, 39.97 dB, 36.63 dB and 39.55 dB, respectively; without introducing the quality loss caused by APS, the PSNR are 29.64 dB, 40.16 dB, 36.64 dB and 39.69 dB, respectively. (a). Bug (Ref.) (b). Mask (Ref.) (c). CS (Ref.) (d). Salmon (Ref.) (e). Bug (AT) (f). Mask (AT) (g). CS (AT) (h). Salmon (AT).

coordinate. The texture filtering unit needs the updated LOD to generate the filtering coefficients as well.

The proposed technique is verified through evaluating the L1 cache performance in terms of the reduction of cache updating activities and the corresponding approximated visual quality. The test models with the corresponding view settings are shown in Fig. 13, where each is rendered to a 512 × 512 viewport. With considering the quality loss induced by the approximated precision shader technique, the average visual quality in PSNR can achieve to 36.4 dB, and Fig. 13 shows the perceptual quality is well-controlled. Fig. 12 summarizes the L1 cache performance with the proposed technique. The proposed GPU architecture has one texture unit and its corresponding L1 cache for each shader cluster. The pixel tasks are almost evenly distributed among the shader clusters, and the evaluation shows the average performance of the four texture L1 caches. Our empirical evaluation reveals 24.57% reduction of L1 cache request in average.

VI. Screen-Space Approximated Lighting

In previous researches, Chang *et al.* [5], [6] propose an approximation technique, which is a pixel duplication scheme, to aggressively reduce the switching power dissipation among shader processors and the corresponding buffers, meanwhile, sacrificing the visual quality. Fig. 14 shows an example of how the pixel duplication scheme operates in a tile-based rasterization architecture. During rasterization, the raster unit checks each rasterized 4 × 4 tile to select only one pixel inside the triangle for executing pixel program as shown in Fig. 14(a). After pixel shading, the shaded color will be directly duplicated to other unshaded pixels in the ROP stage as Fig. 14(b) illustrates.

Based on this design concept, an improved approximation technique is proposed in this work. Since illumination is usually revealed locally smooth on object surfaces, the discontinuous high-frequency illumination occurs near geometric silhouette edges. It is reasonable to approximate the illumination in the forward lighting pass. Therefore, a *tile-based screen-space approximated lighting (SSAL)* technique is proposed to piecewise-linearly approximate or interpolate the un-shaded pixels with the shaded pixels. The degree of approximation relates to the area of a projected triangle, and the corresponding idea is shown in Fig. 15(a). Within the tile-based rasterization process described in Section III, a tile is hierarchically subdivided to 4 × 4 and 2 × 2 screen sub-tiles granularities. The proposed technique, different from the previous approaches [5], [6], examines the four corners of a 4 × 4 sub-tile to determine if the tile is completely covered or not. If a 4 × 4 sub-tile is completely within the triangle, the un-sampled pixels is approximated based on a linear plane equation, $f(x, y) = ax + by + c$, where the coefficients are derived with least square fitting:

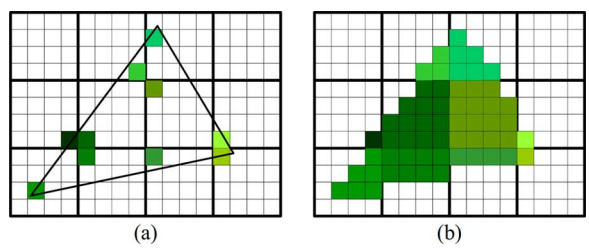

Fig. 14. Pixel duplication scheme proposed by Chang *et al.* [5], [6].

$$\begin{bmatrix} x_0 & y_0 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} I_0 \\ \vdots \\ I_n \end{bmatrix} \tag{6}$$

$$\mathbf{A}\hat{x} \approx \mathbf{b},$$

$$\hat{x} \approx (\mathbf{A}^{\mathrm{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{b} \tag{7}$$

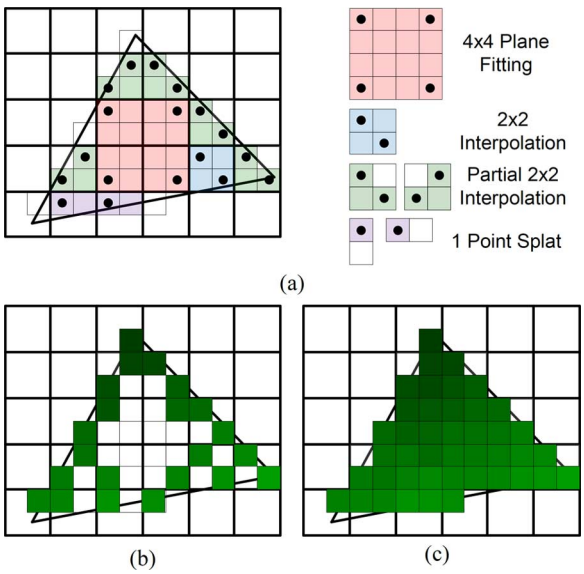

Fig. 15. (a). The sampling pattern of screen-space subdivision test. (b). Pixels for lighting. (c). The approximated results.

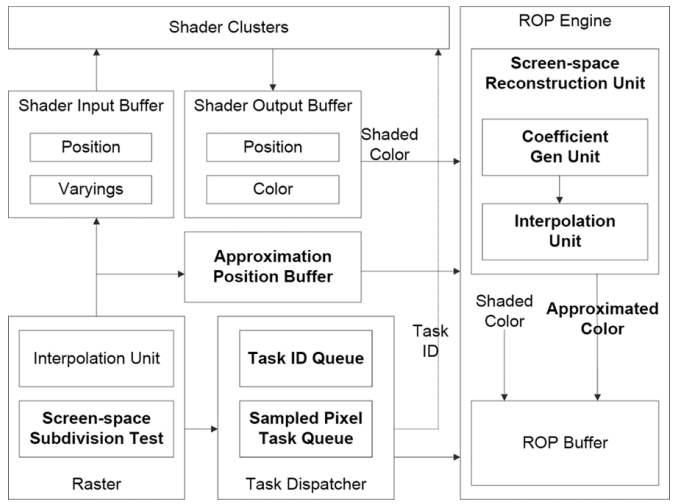

Fig. 16. The proposed screen-space approximated lighting architecture.

where (x_n, y_n) is the position of pixel n and I_n represents the illumination intensity. $\mathbf{A}$ is the matrix composed of shaded pixel positions, and $\mathbf{b}$ comprises the corresponding intensities in (6); $\hat{x}$ is the fitted plane coefficient vector (a, b, c). Since the sampling positions are pre-determined for the 4 × 4 sub-tile approximation (i.e., the four tile corners), the inverse matrix can be established in advance. The other 12 un-shaded pixels can then be approximated by simple linear operations. If the 4 × 4 sub-tile check is not passed, that is, not all four corners are inside the triangle, the approximation further shifts to 2 × 2 granularity. Fig. 15(a) shows the sampling patterns for the full and partial 2 × 2 interpolation scheme, where the two black dots are the sampled pixels for shading, and their shaded results are splatted to neighboring un-shaded pixels. Moreover, the one point splat scheme for a 1 × 2 or 2 × 1 case directly splats the intensity of the shaded pixel to the neighboring un-shaded one. Fig. 15(b) illustrates an example of the sampled pixel for lighting, and Fig. 15(c) is the corresponding approximated results.

The proposed architecture is shown in Fig. 16. The raster engine adopts the *Screen-space Subdivision Test* unit to determine the approximation schemes during the tile traversal process described in Section III. The unit performs simple logic operations for checking specific pixel positions within a tile. After the approximated patterns are decided, the raster will notify the task dispatcher to issue pixel threads, which are needed to be executed, to the shader clusters. The interpolated input variables of the sampled pixels, such as position and *varying*, are stored in the input buffer. In addition, the pixel positions of the approximated pixels are stored into the *Approximation Position Buffer*. Based on the task ID, shader clusters can access the pixel input variables of a pixel program. After shading the sampled pixels, the shader processor stores the corresponding pixel position and color to the output buffer. Next, the task dispatcher triggers the ROP unit to approximate the rest un-shaded pixels. The *Screen-space Reconstruction Unit* collects the shaded pixels and approximates the others. For the 4 × 4 plane fitting pattern, the *Coefficient Gen Unit* generates the set of plane coefficients (i.e., a, b and c) for later interpolation. For the 2 × 2 approximation pattern, the unit simply averages the two shaded pixel color stored for splatting. The rest of one-point splat cases simply reuse the buffered shaded color.

The proposed architecture and approximation flow of the *ROP Engine* are illustrated in Fig. 17. Before the approximation phase, the shaded pixels are dispatched to the shaded pixel color buffer in the *Coefficient Gen Unit*. The *Coefficient Gen Unit* then generates the corresponding plane coefficients and interpolate the un-shaded pixels for the 4 × 4 case. For the other cases, the pixel is approximated based on the 2 × 2 interpolation or one point splatting scheme. Finally, the visibility of the approximated pixel still needs to be examined through the depth and stencil tests before updating the color cache.

Fig. 18 shows the approximated results and the original ones for subjective evaluation. Furthermore, we summarize the approximation quality in Table I. These models are illuminated by a point light source. Considering the quality loss caused by the approximated precision shader cluster, the average PSNR is 43.16 dB for the 768 × 768 viewport setting, and 41.25 dB for the 512 × 512 viewport. Our lighting shader program is similar to Fast TnL, which performs lighting on vertices and rasterize the intensity, including diffuse and specular components. We then multiply the rasterized lighting intensity components with pixel color. The compiled pixel shader program has 7 vector instructions for our SIMD processors. Since the quality degradation is barely perceived due to the high PSNR performance, Fig. 18 simply lists 768 × 768 cases for reference. According to our observation, the average PSNR of 512 × 512 viewport case is smaller than the 768 × 768 case. The possible reason is that, since the SSAL is performed in screen space, the illumination distribution reveals more smooth variation while the projected triangle is larger as the viewport size grows. The piecewise approximation patterns can deliver better approximation quality. Fig. 19(a) illustrates the percentage of power reduction among

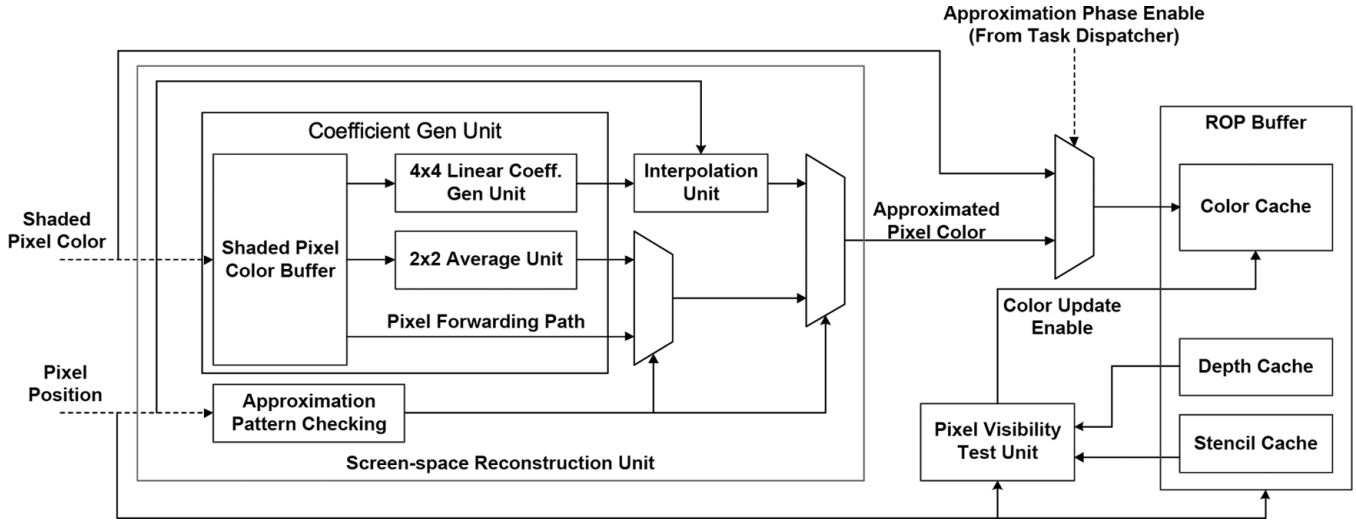

Fig. 17. The detailed architecture of the proposed ROP engine which supports screen-space approximated lighting.

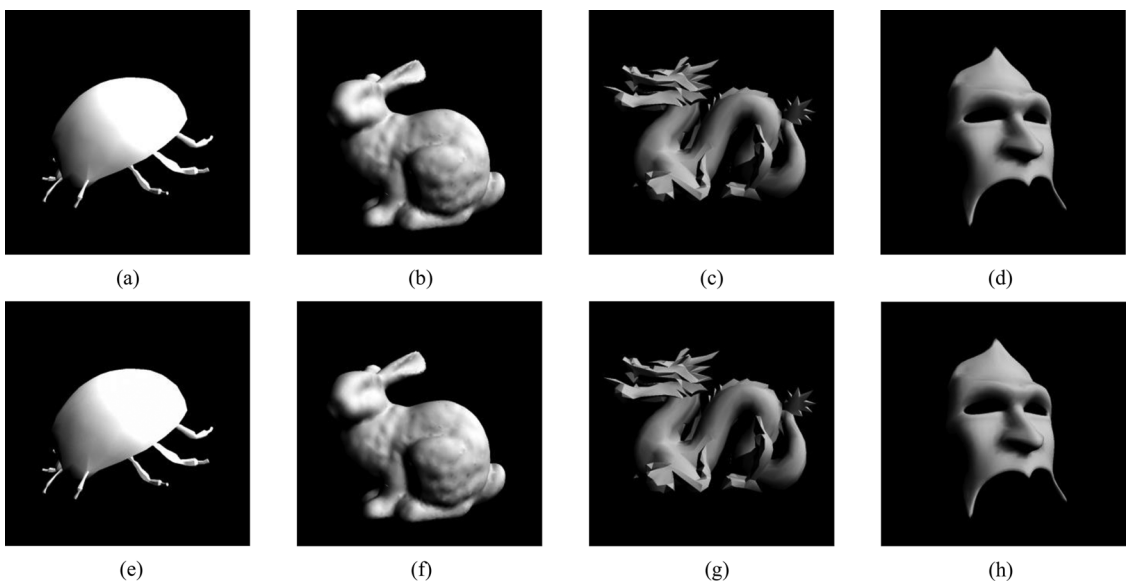

Fig. 18. Screen-space approximated lighting results: From left to right, the triangle numbers of these models are 0.8 K, 13.6 K, 5.1 K and 2 K. From (a) to (d) are the reference lighting results of 768 × 768 viewport case; from (e) to (h) are the corresponding approximated results. The quality degradation caused by the APS technique is considered as well.

the shader clusters and the corresponding memory buffers. Moreover, the power reduction brought by the approximated precision shader (APS) technique is shown as well. It shows that the percentage of power reduction is 47.5% in average, and 52.32% with APS technique. The corresponding approximated quality are 42.3 dB and 42.2 dB in average. Fig. 19(b) shows the quality and performance comparisons between the proposed technique and the work of Chang *et al.* [5], [6], where we only average the lighting cases for fair comparison since the quality of textured cases of [5], [6] degrade seriously by the pixel duplication scheme. In [5], [6], the average visual quality for 2 × 2 approximation is 36.8 dB and 31.4 dB for 4 × 4 case. The corresponding percentage of estimated power reduction are 54% and 75.5% respectively. In terms of stability of visual quality, [5], [6] reveals higher variation. Our proposed architecture can provide a relative high (over 5.4 dB) and stable visual quality.

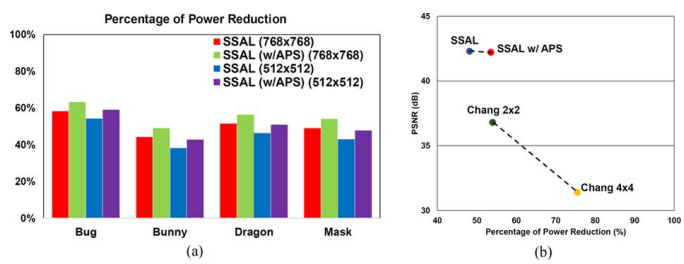

Fig. 19. Performance evaluation of SSAL technique. (a). Percentage of power reduction. (b). Compared with pixel duplication scheme proposed by Chang *et al.* [5], [6].

VII. Implementation Results

The prototype chip is fabricated with TSMC 45 nm technology. The corresponding chip specification and micrograph is shown in Table II and Fig. 20. The chip area is 11.16 mm^2,

TABLE I
SSAL QUALITY SUMMARY UNDER VIEWPORT SIZE AT 768 × 768 AND 512 × 512 CASES

	Bug	Bunny	Dragon	Mask
	768×768			
PSNR (w/ APS)	44.14 dB	42.3 dB	44.7 dB	41.4 dB
PSNR (w/o APS)	44.21 dB	42.37 dB	45 dB	41.5 dB
	512×512			
PSNR (w/ APS)	43.6 dB	39.6 dB	42.2 dB	39.5 dB
PSNR (w/o APS)	43.73 dB	39.66 dB	42.3 dB	39.6 dB

and the core area is 7.84 mm^2. The TD is in short for task dispatcher, DF for data fetch unit. Maximum throughput of arithmetic operation can reach to 33.6 GFLOPS while the working frequency is 350 MHz. The power consumption is 130.3 mW with the workload of activating the whole GPU pipeline including transformation, rasterization and ROP per primitive. Considering the rendering performance, 2.8 G vertices/s and 5.6 G pixels/s can be achieved. In our evaluation, the power reduction ratio of the shader clusters for the *Approximated Precision* architecture is 11.25%, 47.5% with *Screen-space Approximated Lighting* technique, and 52.32% when combining both techniques, as Fig. 21(a) shows. Regarding the introduced interpolation unit and *Coefficient Gen Unit* for SSAL, the average power dissipation are about 0.1% and 0.4% of the whole chip, respectively. The overheads are quite insignificant. Table II also summarizes the specification comparisons. Note that, the arithmetic performance of [5] is higher because the texture unit is designed as a programmable processor named as configurable texture unit (CFU), which is not implemented in this work. Since [7] is not a dedicated design for graphics, we only list it here as a reference. To compare with state-of-the-art related mobile GPUs [4], [5], [7], we adopt Mvertices/s per mW as a performance index, which is derived while all the unified shader processors are computing vertex transformation. The reference comparisons are done with normalizing the process technology for each work for fair comparison, according to $P = CV^2F$. Note that our architecture includes four texture units and a 64 Kbytes texture L2 cache, we exclude the power consumption of two texture units and L2 cache to compare with these works for fair comparison. Compared with these works, our design is 10.93 × over [7], 3.87 × over [5] and 1.5 × over [4] as Fig. 21(b) shows.

VIII. LIMITATIONS AND FUTURE WORKS

In this work, we perform approximation to reduce power consumption on the hardware GPU pipeline from algorithm perspective. There are certain issues worthy of further improving.

1) Aliasing and Temporal Coherency: Since the proposed SSAL is performed per tile basis with certain static sampling patterns, which is limited to the tile-based architecture, the aliasing effect might happen. Moreover, the temporal coherency needs to be maintained for these approaches. In the future, we will focus on improving the approximation quality with better sampling strategy, meanwhile considering the temporal coherency cross multiple frames.

2) Limited Evaluation: Currently, the evaluations are conducted with simple shader programs on an object. Since the proposed techniques are directly performed in the forward hardware pipeline of a GPU, the evaluation may be not complete. However, our purpose is to evaluate the effectiveness of our proposed schemes for lighting and texturing, which may be biased when incorporating multiple shader effects. As for the test scenes, a scene is generally composed of multiple objects, and our evaluation for single object can be superposed since the current PSNR evaluation excludes the unshaded pixels. These proposed techniques can simply be regarded as extensions of the shader effects. In the future, we will explore the possibilities of the proposed techniques to more flexible shader usages.

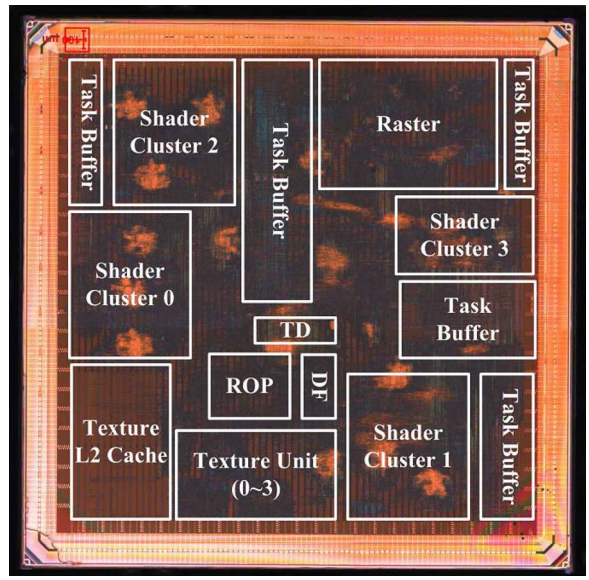

Fig. 20. The fabricated chip micrograph.

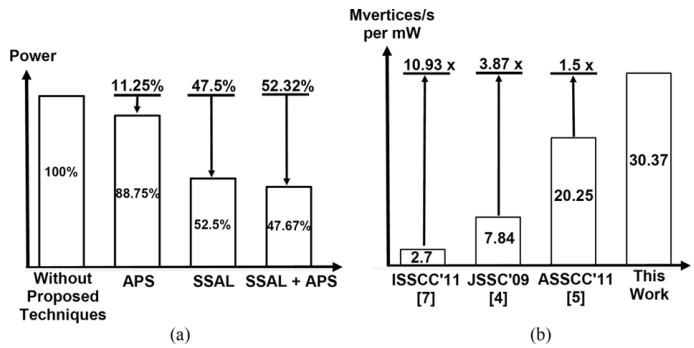

Fig. 21. Performance evaluation. (a). Power reduction ratio with proposed techniques. (b). Performance comparison with state-of-the-art design.

TABLE II
SPECIFICATION COMPARISON AMONG STATE-OF-THE-ART MOBILE GPUS

	ISSCC'11 [7]	JSSC'09 [4]	ASSCC'11 [5]	This Work
Technology	TSMC13	TSMC18	TSMC65	**TSMC45**
Chip Size	16mm² (4x4)	SP 9.7mm² GPU 17.2mm²	16mm² (4x4)	**11.16mm² (3.34 x 3.34)**
Clock Freq.	200MHz	200MHz	300MHz	**350 MHz**
Power	216mW	153mW	172.6mW	**130.3mW**
Arithmetic Performance	1.6GFLOPS	--	43.8GFLOPS	**33.6GFLOPS**
Graphics Performance	25M Polygons/s 1.6G Pixels/s	141M Vertices/s	1.2G Vertices/s 2.4G Pixels/s	**2.8G Vertices/s 5.6G Pixels/s**

3) Thread Scheduling Policy: Furthermore, we will explore the possibility of the scheduling policy among the heterogeneous processor configuration to achieve the multi-level power adjustment through dynamic thread scheduling. The power consumption can be further controlled with finer granularity.

4) Analysis on APS Processor Configuration: Currently, the purpose of APS in this hardware prototype is to verify the level of induced quality degradation while partially lowering the power consumption with the APS processor. Therefore, the configuration is an experimental setting. Moreover, it is difficult to conduct a precise trade-off analysis since the complexity of shader programs varies. To properly program the APS, we also need a dedicated compiler design as Pool *et al.* [16] proposed.

However, we can still estimate the trend of the trade-off. Since an APS cluster can relatively reduce about 45% power consumption compared with a full precision shader cluster, we can linearly scale the percentage of power reduction. While executing approximately 25% pixels, about 11.25% power consumption is saved in the current hardware configuration. Respectively, 22.5%, 33.75% and 45% power consumption might be saved while the number of the APS clusters is increased from 2 to 4, and combined with the full precision shader clusters to form the combinations of a GPU with four shader clusters. Regarding quality, for the texturing case, one set of APS causes 0.1 dB PSNR degradation in average; for the lighting case, one set of APS causes 0.14 dB and 0.1 dB PSNR degradation in average, under the viewport size of 768×768 and 512×512, respectively. The PSNR might not reveal significant degradation while adopting other configurations. Note that the quality evaluation is under our simple test cases. In the future, a complete trade-off analysis will be further explored, which requires amount of empirical evaluations on various shader effects and scene complexities.

5) Multiple-Voltage-Domain Design: Multiple-voltage-domain design is a possible solution to further achieve low power consumption. Since the APS shader cluster is a lightweight architecture, the critical path is shorter than that of the full precision ALU units. The APS shader cluster might be able to be realized with lower supplied voltage while the operating frequency remains the same. It could be our potential future research direction regarding power saving in a hybrid processor architecture.

IX. Conclusions

In this work, a 130.3 mW 16-core mobile GPU is proposed with three power-aware approximation techniques to enhance the energy efficiency. The experimental *Approximated Precision Shader* architecture requires 55% power consumption for executing a pixel program in our empirical evaluation, and the area of the approximated precision shader cluster is 61% compared with a full precision shader cluster. In addition, 24.57% of the texture L1 cache request is reduced with the proposed *Approximated Texturing* technique. Moreover, the *Screen-space Approximated Lighting* technique can save 47.5% of power consumption in processor switching activities, and 52.32% if combined with the *Approximated Precision* technique. These techniques are empirically verified for the effectiveness of power saving and memory dynamics meanwhile maintaining satisfactory approximation visual quality.

Acknowledgment

The authors thank TSMC University Program for supporting the advanced process technology, and Chip Implementation Center (CIC) for EDA tool and measurement services.

References

[1] T. Akenine-Möller and J. Ström, "Graphics for the masses: a hardware rasterization architecture for mobile phones," in *Proc. ACM SIGGRAPH*, 2003, pp. 801–808.

[2] Y.-M. Tsao, C.-H. Chang, Y.-C. Lin, S.-Y. Chien, and L.-G. Chen, "An 8.6 mW 12.5 Mvertices/s 800 MOPS 8.91 mm² stream processor core for mobile graphics and video applications," in *Proc. IEEE Symp. VLSI Circuits*, 2007, pp. 218–219.

[3] Y.-M. Tsao, C.-H. Sun, Y.-C. Lin, K.-H. Lok, C.-J. Hsu, S.-Y. Chien, and L.-G. Chen, "A 26 mW 6.4 GFLOPS multi-core stream processor for mobile multimedia applications," in *Proc. 2008 IEEE Symp. VLSI Circuits*, 2008, pp. 24–25.

[4] B.-G. Nam and H.-J. Yoo, "An embedded stream processor core based on logarithmic arithmetic for a low-power 3-D graphics SoC," *IEEE J. Solid-State Circuits*, vol. 44, no. 5, pp. 1554–1570, 2009.

[5] C.-M. Chang, Y.-J. Chen, Y.-C. Lu, C.-Y. Lin, L.-G. Chen, and S.-Y. Chien, "A 172.6 mW 43.8 GFLOPS energy-efficient scalable eight-core 3D graphics processor for mobile multimedia applications," in *Proc. A-SSCC*, 2011, pp. 405–408.

[6] Y.-J. Chen, J.-C. Su, C.-M. Chang, Y.-C. Lu, and S.-Y. Chien, "Configurable pixel shader workload reduction technique for mobile GPUs," in *Proc. IEEE 1st Global Conf. Consumer Electronics (GCCE)*, 2012, pp. 44–48.

[7] H.-E. Kim, J.-S. Yoon, K.-D. Hwang, Y.-J. Kim, J.-S. Park, and L.-S. Kim, "A 275 mW heterogeneous multimedia processor for IC-stacking on Si-interposer," in *IEEE ISSCC Dig. Tech. Papers*, 2011, pp. 128–130.

[8] H.-E. Kim, J.-S. Yoon, K.-D. Hwang, Y.-J. Kim, J.-S. Park, and L.-S. Kim, "A reconfigurable heterogeneous multimedia processor for IC-stacking on Si-interposer," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 4, pp. 589–604, Apr. 2012.

[9] J. Pineda, "A parallel algorithm for polygon rasterization," *ACM SIGGRAPH Comput. Graph.*, vol. 22, no. 4, pp. 17–20, 1988.

[10] J. McCormack and R. McNamara, "Tiled polygon traversal using half-plane edge functions," in *Proc. ACM SIGGRAPH/EUROGRAPHICS Workshop on Graph. Hardware*, 2000, pp. 15–21.

[11] M. D. McCool, C. Wales, and K. Moule, "Incremental and hierarchical hilbert order edge equation polygon rasterization," in *Proc. ACM SIGGRAPH/EUROGRAPHICS Workshop Graph. Hardware*, 2001, pp. 65–72.

[12] F. C. Crow and B. A. Vignon, "Tile based precision rasterization in a graphics pipeline," U.S. patent 20080024497 A1, Jan. 31, 2008.

[13] F. C. Crow, "Smooth rasterization of polygonal graphics primitives," U.S. Patent 8237738 B1, Aug. 7, 2012.

[14] Z. S. Hakura and A. Gupta, "The design and analysis of a cache architecture for texture mapping," in *Proc. 24th Annu. Int. Symp. Comput. Architecture*, 1997, pp. 108–120.

[15] J. Pool, A. Lastra, and M. Singh, "Energy-precision tradeoffs in mobile graphics processing units," in *Proc. IEEE Int. Conf. Comput. Design (ICCD 2008)*, 2008, pp. 60–67.

[16] J. Pool, A. Lastra, and M. Singh, "Precision selection for energy-efficient pixel shaders," in *Proc. ACM SIGGRAPH Symp. High Performance Graphics*, 2011, pp. 159–168.

[17] *Big.LITTLE Technology: The Future of Mobile*, 2013 [Online]. Available: http://www.arm.com

[18] V. del Barrio, C. Gonzalez, J. Roca, A. Fernandez, and R. Espasa, "ATTILA: a cycle-level execution-driven simulator for modern GPU architectures," in *Proc. 2006 IEEE Int. Symp. Performance Analysis of Systems and Software*, Mar. 2006, pp. 231–241.

[19] L. Williams, "Pyramidal parametrics," *ACM SIGGRAPH Comput. Graph.* vol. 17, no. 3, pp. 1–11, Jul. 1983 [Online]. Available: http://doi.acm.org/10.1145/964967.801126

Yu-Jung Chen received the B.S. degree in electrical engineering from National Cheng Kung University, Tainan, Taiwan, and the M.S. degree in communication engineering from National Taiwan University, Taipei, Taiwan, in 2007 and 2009, respectively.

He is currently pursuing the Ph.D. degree at the Graduate Institute of Electronics Engineering and Department of Electrical Engineering, National Taiwan University. His research interests include GPU architecture, parallel programming, real-time rendering and light transport simulation.

Chao-Hsien Hsu received the B.S. and M.S. degrees in the Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan, in 2003 and 2005, respectively. Since 2013, he has been pursuing the Ph.D. degree in the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan.

His major research interest is mobile GPU architecture.

Chung-Yao Hung received the B.S. degree from the Department of Electrical Engineering and the M.S. degree from the Graduate Institute of Electronics Engineering (GIEE), National Taiwan University (NTU), Taipei, Taiwan, in 2011 and 2013, respectively.

He is currently an engineer with MediaTek, Hsinchu, Taiwan. His research interests include GPU architecture, multimedia systems, and image signal processing.

Chia-Ming Chang received the B.S. degree in computer science from Tamkang University, Taipei, Taiwan, in 1999, and the M.S. and Ph.D. degrees in computer science from National Tsing Hua University, Hsinchu, Taiwan, and the Graduate Institute of Networking and Multimedia (GINM), National Taiwan University, Taipei, Taiwan, in 2001 and 2014, respectively.

He is now with ARM Inc., Taipei, Taiwan. His research interests are computer graphics and GPU hardware architecture design.

Shan-Yi Chuang received the B.S. degree from the Electrophysics Department, National Chiao-Tung University, Hsinchu, Taiwan, in 1998. He is currently pursuing the M.S. degree at the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan.

His research interests include multimedia system and digital signal processing.

Liang-Gee Chen (F'01) received the B.S., M.S., and Ph.D. degrees in electrical engineering from National Cheng Kung University, Tainan, Taiwan, in 1979, 1981, and 1986, respectively.

In 1988, he joined the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan. Currently, he serves as the Chair Professor and received the National Professorship of Taiwan. His research interests include DSP architecture design, video processor design, and video coding systems. He has over 550 publications and 22 U.S. patents. He has conducted more than 100 technology transfers and helped two start-ups IPO successfully. He is the founder of Taiwan IC Design Society, the Chip Implementation Center (CIC), NTU SOC Center, and Graduate Institute of Electronics Engineering. In 2007, he established the Creativity and Entrepreneurship Program at National Taiwan University.

Dr. Chen has served on the Editorial Boards of many IEEE transactions, including IEEE TRANSACTIONS ON VLSI, IEEE TRANSACTIONS ON VIDEO TECHNOLOGY, and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS. He was also invited as the Chair of the technical program committees for 2009 IEEE ICASSP and ISCAS 2012. In 2011, he received the Best Paper Award of IEEE Custom Integrated Circuits Conference (CICC).

Shao-Yi Chien (S'99–M'04) received the B.S. and Ph.D. degrees from the Department of Electrical Engineering, National Taiwan University (NTU), Taipei, Taiwan, in 1999 and 2003, respectively.

During 2003 to 2004, he was a research staff in Quanta Research Institute, Tao Yuan County, Taiwan. In 2004, he joined the Graduate Institute of Electronics Engineering and Department of Electrical Engineering, National Taiwan University, as an Assistant Professor. Since 2012, he has been a Professor. His research interests include computer graphics, real-time image/video processing, video coding, computer vision, and the associated VLSI and processor architectures.

Dr. Chien serves as an Associate Editor for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY and Springer *Circuits, Systems and Signal Processing* (CSSP). He served as an Associate Editor for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I: REGULAR PAPERS in 2012–2013 and served as a Guest Editor for Springer *Journal of Signal Processing Systems* in 2008. He also serves on the technical program committees of several conferences, including ISCAS, ICME, SiPS, A-SSCC, and VLSI-DAT.