

Arithmetic Circuits for Energy-Precision Tradeoffs in Mobile Graphics Processing Units

Jeff Pool, Anselmo Lastra and Montek Singh

***Abstract** — In mobile devices, limiting the Graphics Processing Unit's (GPU's) energy usage is of great importance to extending battery life. This work shows that significant energy savings can be obtained by reducing the precision of graphics computations, yet maintaining acceptable quality of the final rendered image. In particular, we focus on a portion of a typical graphics processor pipeline—the vertex transformation stage—and evaluate the tradeoff between energy efficiency and image fidelity. We first develop circuit-level designs of arithmetic components whose precision can be varied dynamically with fine-grained power gating techniques. Spice simulation is used to characterize each component's energy consumption, based on which a system-level energy model for the entire vertex stage is developed. We then use this energy model in conjunction with a graphics hardware simulator to determine the energy savings for real workloads. Results show that significant energy savings (>60%) can be obtained by lowering the arithmetic precision of this stage without causing any noticeable artifacts in the final image. Furthermore, our approach allows for even greater energy savings for only a modest loss of image quality. Thus, this work finds that the tradeoff between energy efficiency and result quality in a GPU can be exploited to significant benefit with the presented circuit designs.*

1 INTRODUCTION

This paper presents work that supports the use of variable-precision arithmetic in mobile graphics processing units (GPUs). We focus on the first stage of a typical graphics processor pipeline, the vertex transformation stage. Building on previous work [1] in which we developed an error model for this stage, we design variable-precision adders and multipliers, simulate them in Spice, and use their energy characteristics to develop an energy model of the vertex transformation stage. With these energy and error models, we simulate several real-world applications and show that significant energy can be saved with no loss in image quality. Furthermore, even greater savings can be realized with graceful degradation of image quality. Figure 1 illustrates this key idea by showing rendered frames from a popular video game at different precisions. Our prior analytical model suggested that we could realize savings of around 35% [1], but this estimate was much too conservative. Our current work shows that savings of around 60% and greater are possible.

In current graphics applications, a large amount of arithmetic is performed many times a second. For each frame (out of the 30-60 frames per second), millions of triangles are transformed from 3D coordinate space into a 2D space. Each one of these triangles is then enumerated into a set of pixels on the screen; there may be from tens to thousands of pixels per triangle. Each pixel is then, at a minimum, written to a frame buffer memory, but it is usually also subject to another round of tens or hundreds of arithmetic operations to compute per-pixel color and texture. In order to accomplish a feat of such magnitude, separate hardware is utilized for all of this processing: the GPU. The main processing steps in a GPU are shown in Figure 2.

Today's graphics hardware demands large amounts of power. Typical high-end desktop graphics cards not only draw current from the motherboard, but also require multiple rails from a ~800W power supply. It is largely for this reason that graphics applications in mobile devices have been very limited and only able to handle modest polygon and pixel rates. Mobile power supplies simply

cannot supply the energy necessary for high performance graphics for an extended period of time.

There is much demand, however, for greater graphics functionality in mobile devices. As mobile users grow accustomed to having graphical capabilities, they will increasingly expect graphics capabilities similar to those of desktop computers. By reducing the energy used in the GPU, we can enable more advanced graphics in mobile devices and/or increase battery life. Moreover, energy-efficient graphics also has a benefit for desktop computers: power consumption will decrease, thereby lessening the power supply and cooling requirements.

In this work, we have focused on the vertex transformation stage of the graphics pipeline to evaluate the feasibility of the concept of energy-precision tradeoffs. The transformation stage converts vertex coordinates from user-defined floating-point representation to screen-space fixed point numbers. This transformation involves translation, rotation, scaling and perspective projection [2]. In addition, this stage computes the contributions of lights to the illumination at the vertices (again a floating-point computation). We have chosen this stage because it is well-known and fairly standard, so we know how to model its architecture for energy and error analysis. Therefore, our strategy has been to focus on vertex shading in this work, so as to determine the feasibility of our approach, and then to subsequently apply it to other stages which account for even higher power consumption, such as pixel shaders and memories [3].

In order to develop an energy model for the GPU's arithmetic as a function of precision, we design several variable-precision adder and multiplier circuits. We apply fine-grain power gating techniques to turn off the voltage supplied to the lower significant bits of an integer's representation. Note that these variable-precision circuits can potentially be of use outside of the graphics domain as well, such as in low-power DSPs, sensor network nodes, and wireless communication [4]. Simulating our designs yields their energy vs. precision characteristics, which we then use to find the energy performance of the entire vertex transformation stage.

Our approach was successfully validated using a hardware simulator. Several frames of real-world applications (e.g., Doom 3, Quake 4) were rendered at varying precisions, and the image quality and energy consumption were quantified. In the average case, 58% energy savings was possible with little or no errors in resulting images. Further, with acceptable errors that would not significantly affect application usage, even higher (73%) energy savings were obtained.

Our variable-precision hardware design technique is adaptable and scalable to the ranges of needed precisions, which can vary widely from application to application. In addition, due to the choice of power-gating rather than clock- or signal-gating, our designs suppress nearly all the leakage current of the circuitry, which is becoming more and more significant as smaller processes are used.

The specific contributions of this work are as follows:

- Design techniques for dynamically fine-grained precision control in arithmetic hardware,
- Application of these techniques to integer arithmetic addition and multiplication units,
- Simulation of the designed circuits to assess energy savings and acceptability of incurred overheads, and an energy model for the vertex transformation stage, and
- Detailed simulation results of real-world applications to evaluate the tradeoff between energy and precision.

The remainder of this paper is organized as follows. Section 2 details the past work in variable-precision arithmetic and low-power techniques. Section 3 presents our work in designing and simulating variable-precision arithmetic hardware. We then apply these circuits' characteristics in simulating the energy savings gained by reducing the precision in the vertex transformation stage of the graphics pipeline in Section 4. Finally, Section 5 gives a conclusion and directions for future work.

2 RELATED RESEARCH

2.1 *Variable precision techniques*

Varying the precision of computations has been explored in prior work. Hao and Varshney [5] have given a thorough treatment of the sources of errors inherent in reduced precision rendering and the number of bits of precision required for a desired output. Akeley and Su [6] have used an interval arithmetic model to find the minimum triangle separation in eye space to ensure correct occlusion. Chittamuru et al. [7] have suggested variable-wordlength texture-mapping.

Outside of graphics, Alalusi and Victor [8] have shown how a coarse-grained approach to reducing precision in hardware can be applied to integer applications. Tong et al. [9] and Jain [10] have reported similar findings for floating-point applications. Software techniques for reducing the bit-width, and therefore power consumption, of a multiplier have been proposed, such as by Han et al. [11].

Varying the quality of results in favor of faster or more reliable operation has also been examined, from Lafruit et al.'s [12] work on graceful degradation of 3D algorithms, to Yasuura et al.'s [13] treatment of energy-performance tradeoffs in computational systems, to Yeh et al.'s [14][15] exploration of error tolerance in physics-based animation. None of these approaches, however, has systematically explored energy-precision tradeoffs in a graphics pipeline.

2.2 *Circuit designs for low power*

There has also been research directed towards low power arithmetic circuit design. Liu and Furber [16] presented a low power multiplier, while Callaway and Schwartzlander [17] detailed the relationship between power and operand size in CMOS multipliers. Tong et al. [18] suggested a digit-serial multiplier design with three discrete bit-widths, resulting in a linear power savings. Lee et al. [19] proposed a variable-precision constant multiplier that uses shifting in the place of multiplication by powers of 2, realizing an energy savings of between 16% and 56%. Most similar to

our work is that of Huang and Ercegovic, who developed two-dimensional signal gating for variable bit-width multipliers [20][21], realizing up to 66% power savings over a baseline implementation.

Phatak et al. [22] presented a low power adder and included a treatment of the adder's power usage dependent on the number of significant bits. Kwong filed a patent for a variable-precision digital differential adder for use in graphics rendering, but has not reported any power results [23]. Park et al. have proposed a scheme in which energy can be traded for quality, similar to [1], in a DCT algorithm using only three "tradeoff levels" [24].

None of these approaches have the same design characteristics we used. Firstly, our targeted applications need very fine-grained control over the operating precision; thus, coarse-grained designs which allow for, for example, 8, 16, or 24 bits simply do not offer the necessary degree of control. Secondly, we believe that our use of power-gating will offer significant returns when also considering the savings in decreased leakage current. To our knowledge, we are the first to propose designs with both these features.

The VFloat library is meant to address some of these same problems – application-specific precisions, reduced leakage current – but has only been implemented for FPGAs [25]. So, these problems are only solved by actually reprogramming the hardware, which is not possible at runtime. More general-purpose processors are still lacking support for these features, yet are being used for many of the same applications.

There are other low-power techniques as well, such as dynamic voltage scaling [26] and coarse-grained power gating [27], which could be used for energy-efficient graphics. These techniques are orthogonal to our work.

3 HARDWARE IMPLEMENTATION OF FINE-GRAINED DYNAMIC PRECISION CONTROL

With the promising results of our error simulations of variable-precision arithmetic in mobile GPUs

[1] and the applicability of variable-precision hardware to other domains [4][8][11][14][15][24][25], we then took to building hardware that delivers the modeled variable-precision energy savings. We chose three common integer adder designs to modify and looked into different ways to adapt a standard array multiplier for variable-precision operation. The adders we used were a ripple-carry adder, a uniform carry-select adder, and a Brent-Kung adder [28], each with their own strengths and weaknesses. The ripple-carry adder is a simple design that uses very little hardware, but has the longest critical path and therefore the longest propagation delay. The carry-select adder is faster but, depending on the implementation, can use nearly twice as much area. The Brent-Kung adder, although it has the highest area requirements, is the fastest of the three and is easily pipelined, making it a popular and commonly-used design.

Any single component subject to power gating underwent three key modifications. First, the arithmetic logic transistors were supplied with either a virtual power (header switch) or ground (footer switch) signal controlled by sleep transistors driven by an enable signal, rather than actual power or ground rails. This modification allows the power delivery to the element to be cut off, thereby practically eliminating the dynamic power consumption, and also potentially reducing leakage power loss through the circuit. When deciding whether to use either a header or footer switch, we consider the power and timing implications of each [29], as well as the desired output in the disabled state. In the second modification, the outputs were either pulled down (for a header) or pulled up (for a footer switch), depending on the larger context of the element, so that any downstream hardware will see a consistent signal. This both reduces downstream switching and allows for transparent integration with existing hardware; no special treatment of floating signals needs to be considered because the outputs of disabled gates are not floating. Since the state of the output does not need to be preserved when disabled, no extra registers are necessary. Lastly, the logic and gating transistors in the circuit were resized in order to minimize the power or timing

overheads of the modified designs [26][29][30]. Figure 3 shows these changes applied to a standard full adder.

Fine-grained power gating, such as we propose, is subject to problems with ground bounce if large portions of the circuit are switched at one time. Rush-current suppression can be implemented by skewing the switching of portions of the circuit [31]. For our design, we can skew the switching by disallowing very large changes in precision at one time. A possible approach is to have the software driver monitor precision changes and sequence overly large ones as a series of smaller changes.

The operating precision is chosen by setting enable lines to each gated unit. Several approaches are available for correctly setting enable signals. The most straightforward is to drive each gated element based on a symbolic constant in a register. Alternatively, any manner of decoding circuitry can be used to translate a shorter enable code bundled with operands in individual enable/disable signals. The specific technique used will depend heavily on the application and the usage patterns of the unit. It is highly likely, however, that whatever area overheads are incurred by the control circuitry will be shared over several functional units, over an entire ALU, or even over multiple ALUs.

3.1 Modified adder designs

Differences in each of the three adders targeted led to distinct approaches to power gating for each. We chose to explore designs of 24 bit integer adders, which are used in single-precision floating point addition, a common operation in many applications. Past research [4][7][1] has shown that for some target applications, the most readily available savings appear in the first twelve least significant bits of a 24-bit adder, where reduced precision will not have an overly negative impact on application utility. We limit the precision control of our proposed designs to the least significant sixteen bits.

3.1.1 Ripple-carry adder

Of the adder circuits we modified, the most straightforward was the ripple-carry adder, which simply uses one full adder per bit of precision needed by the operands. In order to apply our technique, we

modify each full adder as described previously and shown in Figure 3. Disabling each successive full adder has the effect of reducing the precision by a single bit. The modified design is shown in Figure 4.

3.1.2 *Carry-select adder*

Carry-select adders are slightly more complicated than a simple ripple-carry adder. They employ several banks of smaller ripple-carry adders to make up one full-width adder, each computing two carry paths in parallel. When the carry out signal from one block enters the next, multiplexers select the correct carry path to output to the next stage, and so on. The first ripple-carry block does not have the second carry path, since its carry-in signal is always '0.' It is treated like the modified ripple-carry adder above. The other type of block is made up of two ripple-carry chains in parallel. Applying our technique to these blocks involved gating each parallel pair of full adders as one unit, leading to less power and area overhead than simply using the single full adder approach. Specifically, our tested design was a uniform carry-select adder which uses four blocks of six full adders, with all but the least significant block performing addition in parallel chains. Figure 5 shows the details of a carry-select block with two layers of full adders gated as a single unit.

3.1.3 *Brent-Kung adder*

Lastly, we modified a 24-bit Brent-Kung adder, one of several parallel adder designs. In contrast to the first two designs we explored, which generate a single bit's sum in one functional unit (a full adder), Brent-Kung adders perform addition on a single bit in several stages [28]. Intermediate stages' outputs are used as inputs to later stages of the same bit, as well as later stages of more significant bits. So, in order to freeze the switching activity in the computation of a single bit, it is only necessary to gate the power of the first stage of that specific bit. We used a footer switch to gate this computation in order to tie the outputs high, as they are treated as complementary signals by other signal paths. So, the eventual sums generated will be 0 in the disabled bits, which results in the same functionality as our other adder designs. While it is possible to explicitly gate the subsequent

stages along a bit's computation path, the extra power savings obtained are minimal and do not justify the additional area and speed overheads incurred. The details of these modifications to the first stage can be seen in Figure 6 and are the only modifications necessary for applying our technique to this adder.

3.2 Modified multiplier designs

Integer multipliers are used in many different application domains with similarly varied usage patterns. So, we explored modifying a 24x24-bit array multiplier for variable-precision operation in several ways. A carry-save array multiplier, abstracted in Figure 7, is constructed with a matrix of cells (blue squares) composed of an AND gate, to generate the partial products, and a full adder. The final summation step (dark blue rectangle) of our design was performed with a ripple-carry adder, for simplicity. This adder was not variable precision, in order to fully separate the two designs, though it would certainly make sense to combine our designs in practice. An $n \times n$ multiplier produces $2*n$ product bits, but, in the larger context of a floating-point multiplier, only the high n bits (green squares) are used, while the low n bits (red squares) are ignored.

The full adder of each of these cells is gated in a fashion similar to that shown in Figure 3, but we also designed versions that have separate gating controls for the signals that propagate downwards and those that propagate to higher bits. First, we tested simply suppressing the low-order bits in the operands. Next, we gated the power to just one operand's lower bits, and then the lower bits of both operands. Finally, we adapted a truncation multiplier with correction constant and extended the column truncation to provide variable-precision operation with power gating. Each of the accompanying illustrations represents the gating applied to an 8x8 adder operating at five bits of precision.

3.2.1 Operand bit suppression

Suppressing the data entering the arithmetic units can be done in different ways. In our tests, we assumed bit suppression at the source registers or before; we do not include specialized circuitry for this purpose. Our results, then, will simply show the dynamic power saved. Since there is no power gating performed, the leakage power will not be reduced.

3.2.2 Single operand power gating

Only varying the precision of one operand (the multiplicand) shows that our design allows for handling operands of different precisions. This yields more precise rounding, if necessary, while still achieving significant power savings. For each bit of reduced precision, another diagonal slice of the multiplication matrix can be gated, as shown in Figure 8. Each diagonal slice consists of half a full adder from the lower bit and half a full adder from the higher bit of the slice, so that the bits that would propagate further left are not affected. This mode will also have the lower bound for energy savings in handling operands of different precisions (one operand at full precision).

3.2.3 Double operand power gating

By gating the low-order bits of both operands, even more circuitry is shut down with each bit of reduced precision. As in part 2, a diagonal slice of the partial product matrix is gated for each bit of the multiplicand, while an entire row is gated for each reduced bit of the multiplier. This gating scheme is shown in Figure 9.

3.2.4 Column truncation

A truncation multiplier saves area and power by simply not implementing low-order columns of the partial product generation stage. A correction constant which reasonably handles the average case is added to the carry-in stage of the existing circuitry to correct for the incurred error, but errors can still be large when the generated partial products in a column would all be 0 or 1. We extended the idea of a truncation multiplier [32][33] by applying power gating to entire columns in order to reduce the

operating precision (Figure 10). As more columns are gated, the correction constant (supplied in a similar manner to the precision selection) is changed by software to minimize the average error.

3.3 Simulation setup

We used Spice to simulate the netlists generated by Electric for power and timing figures for a .13 μm TSMC library with a V_{dd} of 1.3V. First, we tested a smaller 8-bit version of each adder exhaustively for correctness, and then we compared the results of adding 200 random operands to a baseline 24-bit ripple-carry adder and visually compared the results to the waveforms produced by the operations in software. We repeated these steps for the multipliers. In this way, we verified the functionality of our designs. The same set of random 24-bit operands was used for the power usage simulations of each modified unit at each operating precision. The current drain through the supply voltage source was measured to determine the power consumed and energy used over these operations. Next, a set of worst-case operands was used to find the longest propagation delay of each adder, measured from the 50% level of the input's voltage swing to the 50% level of the slowest output's voltage swing. Leakage power was found by first performing an operation on random 24-bit operands to approximate the average case current draw. Then, power was measured 500ms after the operation to allow for the dynamic current draw to fade away, leaving only quiescent current. We also devised an experiment to time the worst case delay in enabling/disabling all 16 controllable bits at a time. This will be, in effect, the timing penalty incurred for dynamically changing precisions.

3.4 Circuit design results

We now present the power savings and area timing overheads of our designed circuits from simulation.

3.4.1 Power savings

The overall power consumption for our adder designs as a function of precision is shown in Figure 11. To demonstrate that our design helps suppress leakage power, which is likely to become

increasingly significant as transistor technologies continue to shrink [34], Figure 12 shows the leakage power for each adder circuit as a function of the operating precision. Similar graphs are shown for the results of the modified multiplier power savings in Figures 13 and 14.

The desired linear power savings are very apparent and significant in our proposed adder designs. When using a Brent-Kung adder, for example, reducing the precision by just four bits will cause each operation to use roughly 80% of the energy used by full precision operations. In many applications, the precision can often be reduced by more than just four bits without sacrificing fidelity. For example, in our graphics simulations [1], it has been shown that up to 12 bits can be lost without causing the application to become unusable. This would give energy savings of close to 50% for additions. Savings of this nature are expected, and are used to justify the use of variable-precision arithmetic in some applications [7]. Also, though there was a slight energy overhead in the Brent-Kung adder, this was made back after reducing the precision by only a single bit.

The ripple-carry and carry-select adders do not have any power overheads. This is due to the extra series resistance added by the gating PMOS transistors causing the short-circuit currents to be reduced. These lower currents do incur modest delay penalties, however, further discussed in Section 3.4.3.

There are some expected characteristics of the power/operation vs. precision trends worth noting. Firstly, the ripple-carry adder has an almost perfectly linear slope. This is exactly what was predicted in the first stage of our work, since precisely one full adder per bit is gated. Second, the carry-select adder has two different regions of savings, due to the structure of the design. The first is seen in precisions 24 through 18, which corresponds to the single layer of full adders being gated in succession. After bit 18, at a precision of 17 and below, the savings are more aggressive as two full adders per bit are gated and consume minimal power.

Leakage power consumption (Figure 12) shows analogous trends, as well. Firstly, all the adders show linear savings, as expected. Also, the carry-select adder displays the same dual-slope that was seen in the total power results. Furthermore, while there are some overheads, due to the added transistors, they are overcome with a reduction in precision by only 4-6 bits.

The power savings for the multiplier designs, Figure 13, are even more promising than those of the adders, due to the quadratic hardware complexity of the multipliers.

Just as the adders displayed interesting behavior, the multipliers have trends that warrant remark. The design with the lowest power savings is that with only one gated operand (“X Gating”), which naturally results in linear power savings. Simple operand suppression is more useful, but, as noted previously, does not stop leakage current (see Figure 14), which will be more of a problem when using a smaller technology. Gating both operands (“XY Gating”) performs better than suppression with a similarly inverse quadratic decay, expected from the gating pattern. Using this approach, one must only reduce the precision by 5 bits in order to see a 50% decrease in power consumption. Column gating exhibited even more dramatic power savings, which is to be expected, as roughly half of the multiplier was disabled (or not implemented) from the start. However, it must be noted that the precision is not guaranteed to be exactly the number specified, since the correction constant does not change with operands, only with precision. Errors of one to a few low-order bits must be acceptable when using this scheme, which limits its utility somewhat but gives it the greatest power savings.

The leakage power vs. precision curves, in Figure 14, resemble those of the full power vs. precision curves of Figure 13. While operand suppression does not reduce leakage power, as was expected, the other designs save significant power, and overcome very small power overheads after only one bit of reduced precision. So, if an application can tolerate reduced precision operation, the power savings will be immediately realized

3.4.2 *Area overheads*

The extra area incurred by the gating and control circuitry must be small enough to warrant its use in a mobile circuit. We found that the overheads were very modest, certainly within acceptable limits [30]. Table I shows the overheads, as transistor counts, for each adder type, and Table II shows the same figures for our multiplier's designs. We have not included the area penalty for precision control circuitry, as it is dependent on the implementation chosen. Also, any overhead of the control circuitry would likely be shared among several units; the amortized impact on a single unit, such as an adder, would likely be acceptably small.

Overheads in the on-chip area are not of a degree to prohibit our designs from being used. In an adder, only 76 extra transistors (an increase of 7.1%) are needed to control 16 bits of precision, and the multiplier needs only 2604 extra transistors (an increase of 13%). Control over 16 bits is likely at the upper threshold of bits of precision that can be safely lost without adversely affecting the function of an application that normally operates at 24. Choosing a design that controls fewer than 16 bits will use even less extra hardware, both by reducing the number of gating network transistors needed and also by simplifying the precision control logic. Therefore, this overhead is within reasonable limits and should not pose serious problems to the chip designer.

3.4.3 *Timing overheads*

The proposed variable-precision units incur two delay penalties. The first is the extra time needed for the input signals to propagate through the resized gates to the outputs. The second is the time taken to change operating precisions, or the turn on time. Table III, lists these figures, and compares the propagation delays of the modified and original designs for the new adders, and Table IV reports our findings for the new multiplier designs.

The timing overheads are also reasonable [30]. Firstly, the turn-on time due to precision changing is a cycle or less for each of the modified designs. This means that changing precisions should be

transparent to the designer and end user, and, if necessary, the operating precision can be changed every cycle with no timing overhead. The propagation delay penalty is also quite acceptable, less than 7% at maximum for our adders and less than 4% at maximum for our multipliers. While this overhead is already quite low, in low-power devices, a high clock speed is usually not the primary concern. In fact, the clock may be dynamically slowed to take advantage of lighter workloads. Our techniques are orthogonal to frequency scaling; both can be used on the same circuitry to gain energy savings.

4 GPU ENERGY MODEL AND SIMULATION

Having designed variable-precision arithmetic circuits with promising energy characteristics, we adapt our earlier work into reduced-precision rendering [1] to incorporate these new results to prove their utility. We build an energy model of the vertex transformation stage of a GPU and then use a graphics hardware simulator to provide the necessary workload information to our model.

4.1 *Energy Model*

In this section, we present the energy model we use for estimating the energy spent in rendering a frame. We justify our use of the characteristics of the newly-designed circuits detailed above as the models for addition and multiplication. We then use the energies for these operations to extend the model to more complex operations. The energy spent in a dot product, for example, is the sum of the energies of its constituent additions and multiplications, and likewise for multiply-add and min/max operations. We also show the model used for reciprocal and reciprocal square root operations.

4.1.1 Addition

Floating-point additions are computationally expensive operations consisting of several steps. First, the operands' exponents must be made equal, which requires shifting a mantissa. Then, mantissas are added. Another normalization step is needed to align the sum's mantissa, and then a configurable rounding step.

To model the energy spent in an addition, we focus on the mantissa addition. Jain [10] found that the energy consumption of a floating-point adder is highly dependent on the width of the significand. In practice, most FPUs on graphics hardware are not fully IEEE compliant, especially in rounding modes supported, since in graphical applications, as Tong et al. [9] noted, "delicate rounding modes are not required." We assume here that simple Round toward Zero (truncation) is used. Shifting the operands can also consume energy, but we consider the following as justification for focusing on the integer adder. First, Sheikh et al. found that the sum of energy consumed in all shifts performed was roughly equal to the energy in the addition stage [35]. However, this was only true for a shifting stage that also calculated necessary information for full rounding modes, such as the sticky bit. Replacing this circuitry with a simpler shifter and limiting the rounding to truncation will, as Jain [10] found, significantly reduce energy consumption. Second, the energy used by the shifter will also likely scale with the width of the operands [8]. So, any energy in the shifter will be a small amount linearly related to the energy in the addition stage.

So, we model the energy used by a floating point adder as an integer adder with the understanding that our estimated energy will be less than the real energy by a small constant factor. We use the results of our new Brent-Kung adder design as our energy model for addition, E_{Add} , as a function of precision p :

$$E_{Add}(p) = BKEnergy[p] \quad (1)$$

4.1.2 Multiplication

Multiplication is modeled as integer multiplication at a given precision. Tong et al. [9] found that 81.2% of the energy consumed in floating point multiplication is spent in the mantissa multiplication or over 98% when the rounding unit is disregarded, which is the case for simple truncation. Therefore, we focus on the mantissa multiplication, and use the results of a standard array multiplier, modified for variable-precision operation with XY operand gating, as presented in Section 3.2.3 as our energy model for multiplication:

$$E_{Mul}(p) = ArrayXYEnergy[p] \quad (2)$$

4.1.3 Reciprocal/reciprocal square root

Historically, several types of iterative reciprocal and reciprocal square root calculations have been used in hardware. SRT division converges upon the correct result linearly, while Newton-Raphson (and others based on Newton's method)[36] and Goldschmidt (and other power series expansions [37]) converge quadratically to the result.

In order to make use of our variable-precision designs, we chose to model reciprocal and reciprocal square root computations with narrow bit-width multiplications introduced by Ercegovic et al. [32], based on Taylor series expansion. This method consists of a reduction step, evaluation step, and post-processing. Several iterations of the evaluation step are needed, for which some operations require only $p/4$ bits of precision. When the energies for all stages are summed, the total consumptions are as follows for a reciprocal (3) and a reciprocal square root (4) operation:

$$E_{RCP}(p) = \log_2 p * \left[5 * E_{Mul} \left(\frac{p}{4} \right) + E_{Add} \left(\frac{p}{4} \right) \right] + E_{Mul}(p) \quad (3)$$

$$E_{RSQ}(p) = \log_2 p * \left[4 * E_{Mul} \left(\frac{p}{4} \right) + E_{Add} \left(\frac{p}{4} \right) + E_{Mul}(p) \right] + E_{Mul}(p) \quad (4)$$

4.2 *Variable-precision GPU simulation results*

In order to utilize our energy model and visualize the actual errors in rendering, we incorporate the model into ATTILA, a cycle-accurate GPU simulator [38]. We added energy logging functionality: when an operation is executed, it calculates its own energy usage based on the current precision (kept constant during a given simulation) and logs this information for further analysis. In addition, the simulator was modified to provide full support for variable-precision vertex transformations. The GPU's arithmetic functions were modified to operate on a custom data type that performs truncation of floating-point results to any specified precision. For each simulated frame, the transformed vertices and resulting color buffer are also saved, which allows for examination of error caused by reducing transformation precision. Thus, the data gathered from the simulator is the energy usage of each operation per cycle, the final frame buffer for the simulated frames, and the positions of transformed vertices.

The applications that we simulated and analyzed were Doom 3, Prey, Quake 4, and a simple torus viewer, all traces released by the ATTILA group specifically for use with the simulator and seen in Figure 15. Several hundred frames (to create useful videos) of the first two applications were simulated, and several sample frames, used for energy and error analysis, were logged for all four applications. We simulated these applications at a resolution of 640 x 480 pixels, which is a higher resolution than all but the newest mobile devices. We also see that relative error is independent of screen size, so the visual errors will not be any more noticeable at higher resolutions; our approach will still apply to newer devices.

4.2.1 *Energy savings*

The energy characteristics of the applications were as generally expected, given our energy model: the energy usage was higher at higher precisions, and decayed quadratically (due to the multiplication unit's savings) towards lower precisions. The energy savings compared to the unmodified circuitry

(far-right data point of each curve “Full”) is significant, even for the variable-precision circuitry running at full precision (“24”), due to the ability to perform intermediate computations of RCP/RSQ operations with less precision. Full-precision hardware does not have this immediate savings. Furthermore, work involved in transforming vertices is not dependent on screen size, so the results were identical for the same frame of a given application at different sizes. Fig. 16 shows the graph of simulated power vs. precision.

4.2.2 *Energy-precision tradeoff*

In prior work, we analyzed the errors incurred by reducing the precision of the vertex transformation stage [1]. Using this analysis with our new energy results, we can now see how much energy could be saved by using the circuits in real applications. Figure 17 shows the energy-precision tradeoff curves for all simulated applications. Energy usage is normalized with respect to the unmodified hardware designs for each application so savings are readily apparent as a percentage of the total energy consumed. We have seen that xy errors did not cause any perceptible errors when these errors were less than a tenth of a pixel on average. Furthermore, applications did not become unusable until the errors in x and y exceeded, on average, a pixel. At these errors, energy saved was roughly 75% and 85%, respectively. However, actual savings were not quite this pronounced, since z -fighting limited the utility of the applications before xy errors grew to an unacceptable level [1].

5 CONCLUSIONS AND FUTURE WORK

We have applied power-gating techniques to several standard adders and an array multiplier, converting them to be dynamic, fine-grained variable-precision circuits. Our designs show significant savings when reducing the precision of integer adders and multipliers in order to save dynamic and static power consumption. We have shown that the overheads caused by this power gating are not expensive, and that the precision only needs to be reduced by one or two bits in order to start seeing the rewards of power savings.

We then used the energy characteristics of the proposed circuits to build an energy model of the vertex transformation stage of a GPU. When we reduce the precision of the arithmetic to a level which will not cause any visible errors, we can save roughly 60% of the energy in this stage. Furthermore, reducing the precision to a degree which causes only minor visual artifacts can save nearly 80% of the energy.

Our results show that significant energy savings can be achieved by using reduced-precision vertex computation. While this work focuses only on the vertex stage, the energy-precision tradeoff proposed here could likely be extended to other parts of the pipeline that also consume significant amounts of energy, such as memory, texture units, rasterization, and the fragment shader. In work derivative from [3], we have estimated the vertex shader to consume around 15-20% of the total energy in a mobile GPU for recent applications; however, the pixel shader, another candidate for variable-precision operation (half-precision operations are already supported), consumes roughly 50% of the energy. We also believe that we can save energy in the memory accesses, which currently constitute around 20% of the total energy, by compressing the smaller reduced-precision data.

We will also extend our circuit designs into a full variable-precision ALU for use in many contexts. This will necessitate designing other functional units, such as shifters and dividers. Next, we will incorporate these circuits into a variable-precision floating-point unit, as there are demonstrated uses for such a unit. With such variable-precision units available, we can explore new approaches to variable-precision data storage and signaling. While we have presented several adder designs, we are confident our approach will apply to other adders, as well, including carry-save adders, Kogge-Stone and other parallel adders, and pipelined implementations. Likewise, we believe we can apply our techniques to different multiplier designs, such as Wallace or Dadda trees. One of these currently unexplored approaches may prove to be most useful in constructing our eventual larger units.

REFERENCES

- [1] J. Pool, A. Lastra, and M. Singh. Energy-Precision Tradeoffs in Mobile Graphics Processing Units. Proceedings of the IEEE International Conference on Computer Design, (2008), October 12-15, Lake Tahoe, California.
- [2] J. D. Foley, A. vanDam, S. K. Feiner, and J. F. Hughes, Computer Graphics: Principles and Practice in C, Addison-Wesley, 2nd edition (1995).
- [3] J. Pool, A. Lastra, and M. Singh. An Energy Model for Graphics Processing Units. Proceedings of the IEEE International Conference on Compute Design, (2010), October 3-6, Amsterdam, The Netherlands.
- [4] Yoshizawa, S., and Miyanaga, Y., Tunable Wordlength Architecture for Low Power Wireless FDM Demodulator, ISCAS, (2006).
- [5] X. Hao and A. Varshney, Variable Precision Rendering, Proceedings of the 2001 Symposium on Interactive 3D Graphics, (2001), pp 149-158.
- [6] K. Akeley and J. Su, Minimum Triangle Separation for Correct Z-buffer Occlusion,” Graphics Hardware, (2006), pp.27-30.
- [7] Chittamuru, J., Burleson, W., and Euh, J., Dynamic Wordlength Variation for Low-Power 3D Graphics Texture Mapping, IEEE Workshop on Signal Processing Systems, (2003).
- [8] S. Alalusi and B. Victor, Variable Word Width Computation for Low Power,” CS 252 Computer Architecture. Berkeley (2000).

- [9] J.Y.F. Tong, D. Nagle, R. Rutenbar, Reducing Power by Optimizing the Necessary Precision/Range of Floating-point Arithmetic. IEEE Trans. on VLSI Systems, pp. 273-286, June, **(2000)**.
- [10] S. Jain, Low-power Single-precision IEEE Floating-point Unit, MIT (Master's Thesis), May, **(2003)**.
- [11] Kyungtae Han; Evans, B.L.; Swartzlander, E.E., Jr., Data Wordlength Reduction for Low-power Signal Processing Software, SIPS, pp. 343-348, October 13-15, **(2004)**.
- [12] G. Lafruit, L. Nuchtegaele, K Denolj, J. Bormuns, 3D Computational Graceful Degradation. IEEE International Symposium on Circuits and Systems, May, **(2000)**.
- [13] H. Yasuura, T. Ishihara, M. Muroyama, in Essential Issues in SOC Design, Youn-Long Steve Lin, Editor, **(2006)**.
- [14] T. Yeh, G. Reinman, S. Patel, P. Faloutsos. Fool Me Twice: Exploring and Exploiting Error Tolerance in Physics-Based Animation, ACM Transactions on Graphics, **(2007)**.
- [15] T. Yeh, P. Faloutsos, M.D. Ercegovac, S. Patel, G. Reinman. The Art of Deception: Adaptive Precision Reduction for Area Efficient Physics Acceleration, 40th Annual IEEE/ACM International Symposium on Microarchitectures, pp.394-406, **(2007)**.
- [16] Y. Liu, S. Furber, The Design of a Low Power Asynchronous Multiplier," International Symposium on Low Power Electronics and Design, pp. 365-371, August, **(2004)**.
- [17] T. K. Callaway, E. E. Swartzlander Jr, Power-Delay Characteristics of CMOS Multipliers, Proceedings of the 13th IEEE Symposium on Computer Arithmetic, Asilomar, California, **(1997)**, July 6-9.

- [18] Y. Tong, R. Rutenbar, and D. Nagle. Minimizing Floating-Point Power Dissipation via Bit-Width Reduction, ISCA, **(1998)**.
- [19] Young-Geun Lee; Han-Sam Jung; Ki-Seok Chung, Low Power Constant Multiplier with Variable Precision Computing Capability, Solid-State and Integrated-Circuit Technology, October, **(2008)**.
- [20] Z. Huang and M. Ercegovac, Two-dimensional Signal Gating for Low-Power Array Multiplier Design, IEEE International Symposium on Circuits and Systems, **(2002)**.
- [21] Z. Huang and M. Ercegovac, Two-dimensional Signal Gating for Low Power in High-Performance Multipliers, Proc. SPIE on Advanced Signal Processing Algorithms, Architectures, and Implementations XII, pp. 499-509, **(2003)**.
- [22] D. Phatak, S. Kahle, H. Kim, J. Lue, Hybrid Signed-digit Representation for Low Power Arithmetic Circuits, Proceedings of the Lower Power Workshop in Conjunction with ISCA, June **(1998)**.
- [23] M. Kwong, Low Power, Variable Precision DDA for 3D Graphics Applications, United States Patent no. 6947056, Sep. 20, **(2005)**.
- [24] Jongsun Park; Jung Hwan Choi; Roy, K., Dynamic Bit-Width Adaptation in DCT: An Approach to Trade Off Image Quality and Computation Energy, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.18, no.5, pp.787-793, May **(2010)**.
- [25] Wang, X. and Leeser, M. 2010. VFloat: A Variable Precision Fixed- and Floating-point Library for Reconfigurable Hardware. ACM Trans. Reconfig. Technol. Syst. 3, 3, Article 16, September **(2010)**.

- [26] J. Mao; Q. Zhao; C., C.G., Optimal Dynamic Voltage Scaling in Power-limited Systems with Real-time Constraints, 43rd IEEE Conference on Decision and Control, December, (2004), pp. 1472-1477.
- [27] M. H. Chowdhury, J. Gjanci, P. Khaled, Innovative Power Gating for Leakage Reduction, IEEE International Symposium on Circuits and Systems 2008, May, (2008), pp. 1568-1571.
- [28] Brent, R. and Kung, H., A Regular Layout for Parallel Adders, IEEE Trans. On Computers, (1982).
- [29] Shi, K., and David Howard, D., Sleep Transistor Design and Implementation – Simple Concepts Yet Challenges to be Optimum International Symposium on VLSI Design, Automation and Testing, (2006).
- [30] Sathanur, A.; Calimera, A., Pullini, A., Benini, L., Macii, A., Macii, E., Poncino, M., On Quantifying the Figures of Merit of Power-Gating for Leakage Power Minimization in Nanometer CMOS Circuits, IEEE ISCAS, (2008).
- [31] Usami, K.; Shirai, T.; Hashida, T.; Masuda, H.; Takeda, S.; Nakata, M.; Seki, N.; Amano, H.; Namiki, M.; Imai, M.; Kondo, M.; and Nakamura, H., Design and Implementation of Fine-Grain Power Gating with Ground Bounce Suppression, 22nd International Conference on VLSI Design, pp.381-386, (2009), January 5-9.
- [32] M. Ercegovic, T. Lang, J-M. Muller, and A. Tisserand, Reciprocaton, Square Root, Inverse Square Root, and Some Elementary Functions Using Small Multipliers, Research Report #97-47, Laboratoire de l'Informatique due Parallelisme, November, (1997).
- [33] Schulte, M. and Swartzlander, E., Truncated Multiplier with Correction Constant, Workshop on VLSI Signal Processing, (1993).

- [34] Roy, K., Mukhopadhyay, S., and Mahmoodi-Meimand, H., Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits, Proceedings of the IEEE, Vol. 91, No.2, February, **(2003)**.
- [35] Sheikh, B.R.; Manohar, R., An Operand-Optimized Asynchronous IEEE 754 Double-Precision Floating-Point Adder, IEEE Symposium on Asynchronous Circuits and Systems (ASYNC), pp.151-162, **(2010)**, May 3-6.
- [36] D. Chen, B. Zhou, Z. Guo, P. Nilsson, Design and Implementation of Reciprocal Unit, 4th Midwest Symposium on Circuits and Systems, pp.1318-1321, Vol. 2, August, **(2005)**.
- [37] N. Foskett, R. Prevett, S. Treichler, Method and System for Performing Pipelined Reciprocal and Reciprocal Square Root Operations, U.S. Patent 7117238, NVIDIA Corporation, October, **(2006)**.
- [38] V. Moya, C. Gonzalez, J. Roca, A. Fernandez, R. Espasa, ATTILA: A Cycle-level Execution-driven Simulator for Modern GPU Architectures, International Symposium on Performance Analysis of Systems and Software, March, **(2006)**.

FIGURES AND TABLES

a) Full precision (24 bits per mantissa)



b) 19 bits per mantissa



(c) 16 bits per mantissa



Figure 1. Frame from the video game Doom 3 simulated at (a) full floating-point precision (24 bits per mantissa), (b) 19 bits, and (c) 16 bits of precision. The first two images are nearly visually identical, yet (b) saved 62% of the energy in the vertex transformation stage. While (c) exhibits a slight degradation in image quality, it saved 76% of the energy.

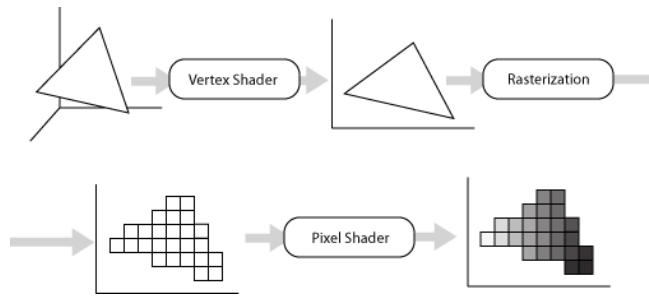


Figure 2. The three stages of a typical graphics pipeline: the vertex shader (performs the vertex transformation), rasterization (generates pixels touched by a triangle), and the pixel shader (performs per-pixel operations).

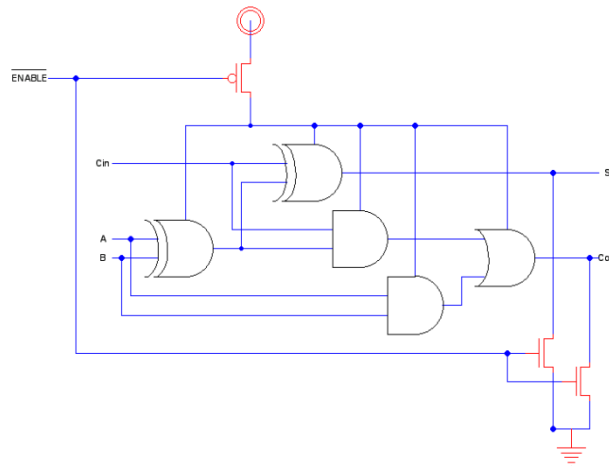


Figure 3. Power gating applied to a full adder. Depending on the value supplied on the Enable line, the transistors in the gates either receive an actual V_{dd} or just a floating input, which does not provide a path for current to follow. The transistors connected to the outputs only pull the values low if the block is disabled, providing components downstream from the adder with a constant value.

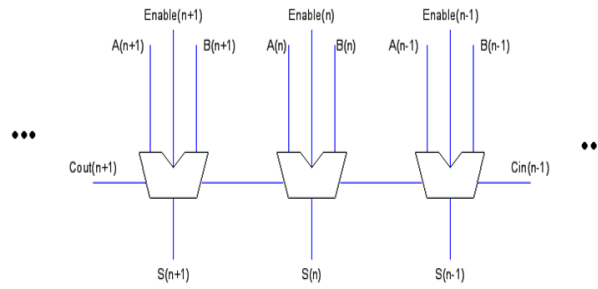


Figure 4. A section of a modified ripple-carry adder. Each full adder has its own "Enable" signal in order to gate the power used by the unit. It is assumed that if Enable(n) is low, then Enable(i) is also low, for all $i < n$.

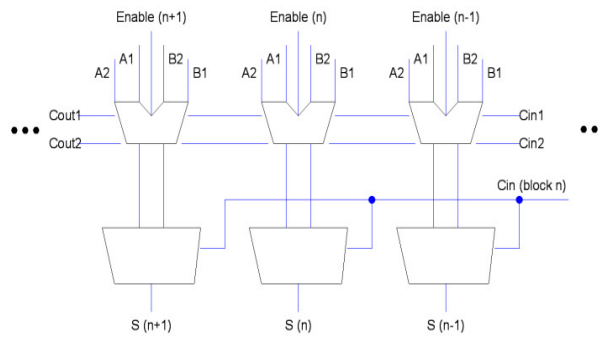


Figure 5. A portion of the double full adder chain of a carry-select adder block. Each gated unit is two modified full adders, as in Figure 10, which share the same gating transistor, saving area and timing penalties. The final sum is chosen with a multiplexer, driven by the carry-in of the previous block.

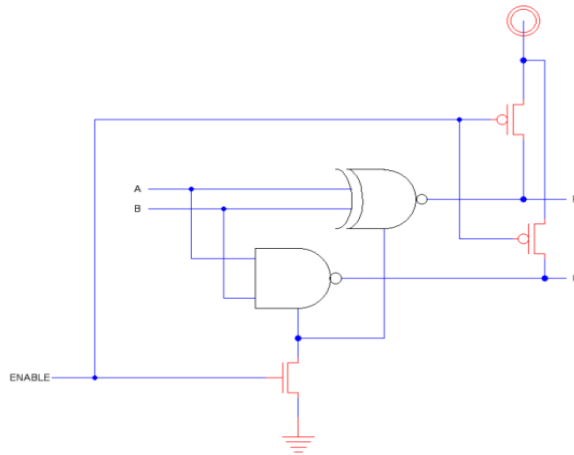


Figure 6. Power gating applied to the first stage of a Brent-Kung adder, the carry generation and propagation signal generation stage. Note the use of the NMOS to supply a virtual ground to the logic gates, and the PMOS to tie the output signals to a logical '1,' characteristics of a footer switch. The outputs are sent further down the computation chain of the current bit, as well as to the next stage of the next significant bit, as complementary signals.

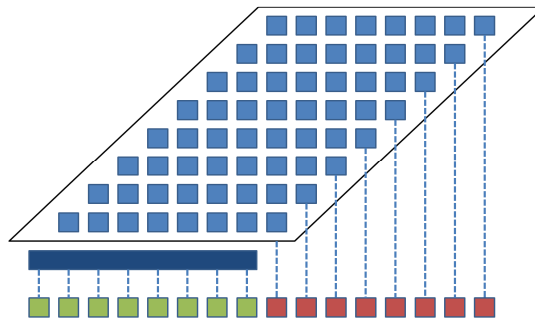


Figure 7. An abstracted representation of an 8x8 carry-save array multiplier, showing partial product generation (blue squares), final adder (dark blue rectangle), used product bits (green squares), and ignored product bits (red squares).

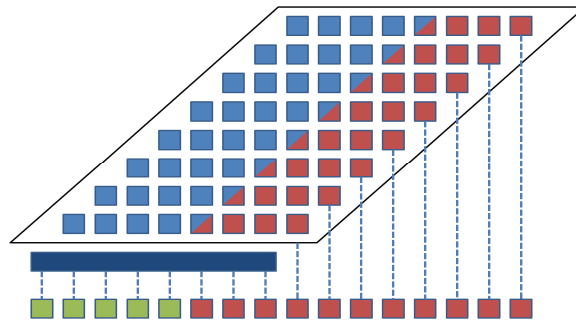


Figure 6. When gating only one operand, the multiplicand, diagonal slices of the partial product matrix are disabled. This allows for more precise rounding, if required.

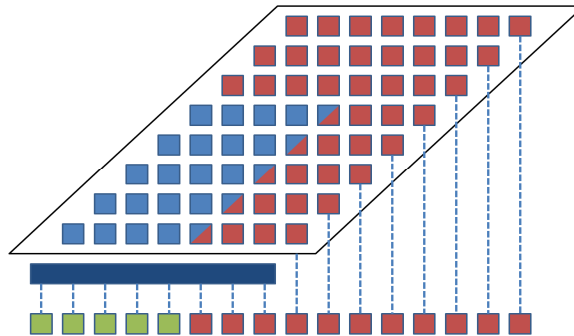


Figure 7. When gating both operands, entire rows of the multiplier's partial product matrix are disabled in addition to the diagonal slices of the multiplicand.

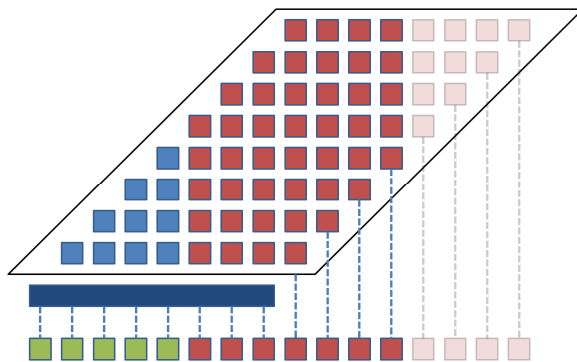


Figure 10. Column truncation extends the premise of a truncation multiplier by applying power gating to entire columns at a time. In addition, not every column is implemented in hardware, saving significant circuit area.

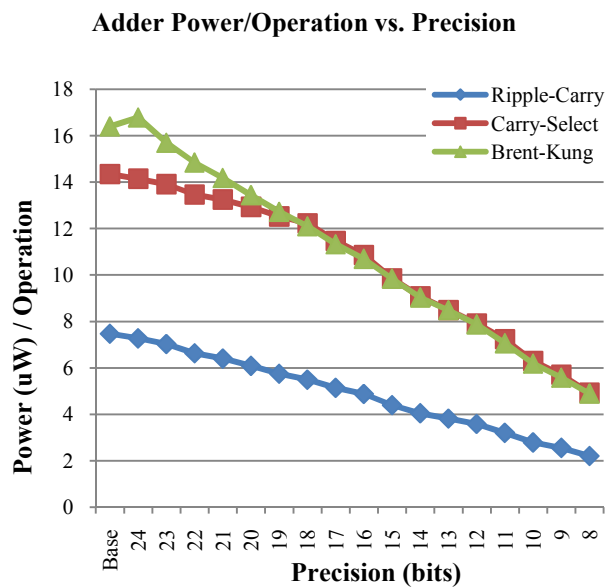


Figure 11. Power/operation vs. precision of the different adder designs. The ripple-carry adder uses very little power per operation, while the carry-select and Brent-Kung use nearly double this amount. These others, though, are significantly faster.

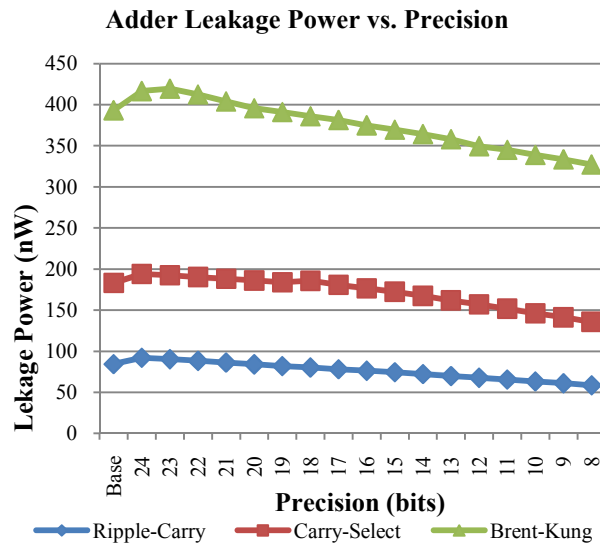


Figure 12. Leakage power vs. precision for each adder. Trends similar to those in the total power are seen in the leakage power figures

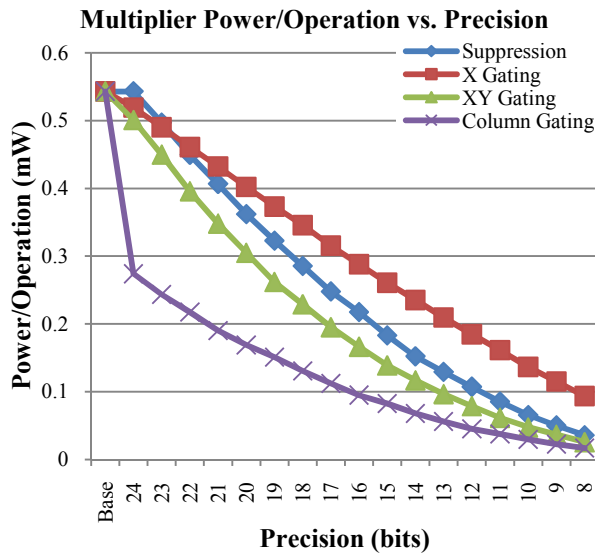


Figure 13. Power/operation vs. precision of the different multiplier designs. Simply gating one operand (“X Gating”) leads to a linear savings, while gating both operands and taking advantage of the multiplier’s quadratic complexity yields more aggressive savings with minimally reduced precision.

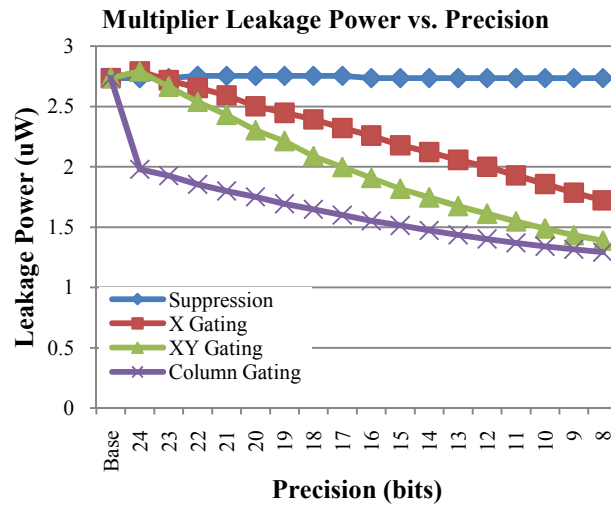


Figure 8. Leakage power vs. precision of the different multiplier designs. Suppressing operand data does not reduce leakage power at all, but the other curves show the same trends observed in the full power savings above.

TABLE I
EXTRA TRANSISTORS NEEDED FOR MODIFIED ADDERS

<i>Adder Type</i>	Transistor count		Increase (%)
	Unmodified	Modified	
<i>Ripple-Carry</i>	672	752	11.9
<i>Carry-Select</i>	1428	1528	7.0
<i>Brent-Kung</i>	2042	2090	2.35

TABLE II
EXTRA TRANSISTORS NEEDED
FOR MODIFIED MULTIPLIERS

<i>Gating Type</i>	Transistor count		Increase (%)
	Unmodified	Modified	
<i>X</i>	20256	23712	17
<i>XY</i>	20256	23712	17
<i>Column</i>	10872	11772	8

TABLE III
TIME OVERHEADS OF THE MODIFIED ADDERS

<i>Adder Type</i>	Critical Path Delay (ns)			Turn-on Time (ns)
	Unmodified	Modified	Increase (%)	
<i>Ripple-Carry</i>	5.6	5.9	6.9	2.1
<i>Carry-Select</i>	2.4	2.5	6.9	1.4
<i>Brent-Kung</i>	1.066	1.069	0.4	1.069

TABLE IV
TIME OVERHEADS OF THE MODIFIED MULTIPLIERS

<i>Gating Type</i>	Critical Path Delay (ns)			Turn-on Time (ns)
	Unmodified	Modified	Increase (%)	
<i>X</i>	6.99	7.26	3.8	7.15
<i>Y</i>	6.99	7.26	3.8	7.15
<i>Column</i>	6.99	7.26	3.8	7.15

a) Quake 4



b) Prey



c) Torus

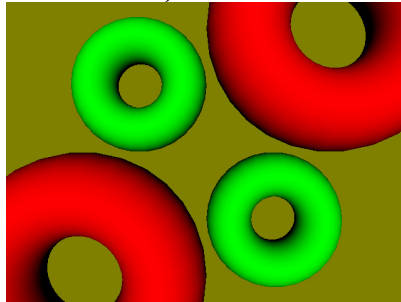
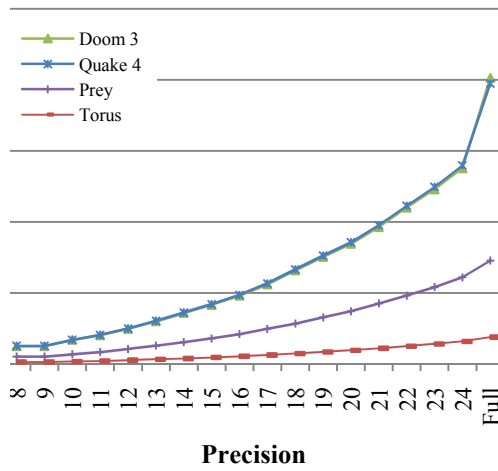
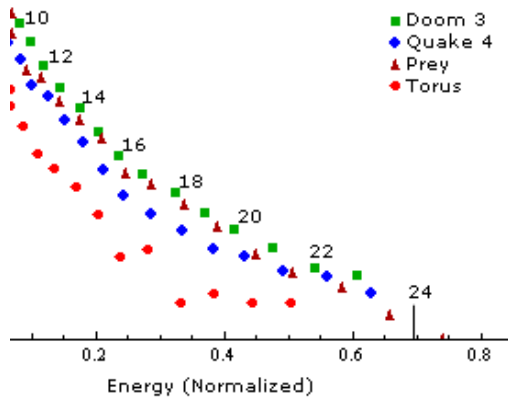


Figure 9. Single frames simulated for error/energy purposes of (a) Quake 4, (b) Prey, and (c) a torus viewer, which has much simpler and more compact geometry.



6. Power consumption as a function of precision, which shows the expected convergence to zero. “Full” precision is the consumption of the unmodified, full-precision circuitry.



7. Energy-Precision tradeoff curves for the four games at 640x480 pixels (note the error on the Error axis). At the far left of the plot is the data point for 8 bits of precision, and by one to the right, with 24 bits of precision (0 error) not visible. For clarity and the inset numbers denote precisions for