Received September 25, 2014, accepted October 10, 2014, date of publication October 15, 2014, date of current version October 20, 2014.

Digital Object Identifier 10.1109/ACCESS.2014.2363113

Processing Time Analysis and Estimation for 3-D Applications

JOSEPH A. ISSA
Department of Electrical and Computer Engineering, Notre Dame University, Zouk Mosbeh 0961, Lebanon
Corresponding author: J. A. Issa (joseph.issa@ndu.edu.lb)

ABSTRACT The performance analysis of a graphics processing unit (GPU) is important for analyzing and fine tuning current and future graphics processors as well as for comparing the performance of different architectures. In this paper, we present an analytical model to calculate the total time it takes for a GPU to retire one frame on a given benchmark. The model also estimates the total retirement time for the same frame on a different GPU using regression estimation model. The model consists of two stages. The first stage entails establishing the measured baseline for a specific frame on a given graphics card, and the second stage entails adjusting the measured baseline and estimating the time it takes to process all draw calls for the same frame on a different graphics card. The model considers the impact of pipeline bottlenecks to process a specific frame, estimates the minimum time it takes to process that frame, and reparameterize the baseline for a different graphics card to calculate new frame retirement times at two different memory frequencies. We used Amdahl's law model to estimate frame retirement time for a different graphics card at higher memory frequencies based on the new adjusted measured baseline with error margin is $<5\%$.

INDEX TERMS 3D benchmarks, graphics processing unit.

I. INTRODUCTION

In recent years, the high demand for gaming and other 3D simulation applications led to development of high performance Graphics Processor Unit (GPU) to keep up with the computation demand behind these application. In particular, the performance of the GPU became an important marketing and engineering factor. The performance analysis of a GPU is a key research topic for many academic and industry groups. It is important to estimate and fine tuning current and future graphics processors as well as comparing the performance of different technologies. This performance analysis enables GPU designers and 3D benchmark software developers to understand the bottlenecks associated with processing a specific frame so that they can fine tune their design and optimize the benchmarks. There are still many challenges to be solved in order to enable GPU engineers to facilitate efficient usage of GPU features to achieve higher performance. Many published papers related to GPU performance analysis are based on simulated results or focus on one specific GPU area that impacts performance. In this paper, we present an analytical model to analyze and estimate frame retirement time on a given GPU for a specific frame. The model is also designed to project the frame retirement time for a different GPU. In particular, we discuss the derivation steps for a graphics pipeline modeling and projection method. The module identifies pipeline bottleneck stages to process one frame for a specific benchmark on a given GPU and also projects frame retirement time for the same frame for a different GPU. The goal is to project the time it takes to process all draw calls for a specific frame as well as project the frame retirement time for the same frame for a different GPU.

This paper extends our previous work presented in [13] by introducing a pipeline analysis and projection for frame retirement time instead of just projecting Frames per Second (FPS) for another GPU using the same benchmark. The process for developing the module is divided into two steps. The first step is to establish a measured baseline by taking certain timing measurements. For this step, we used Nvidia GeForce 9800GX2 [4]. These measurements are used in turn to construct and parameterize baseline for these graphics cards. Once a baseline is constructed, it can be extended to a model for a targeted graphics card by revising its baseline parameter values. The graphics card for which we will be projecting is Nvidia NV280 [5]. The benchmark used for our analysis is 3DMarkVantage [17] benchmark GT4 frame #2353 (GT4:2353).

Our analysis and projection model entails the following steps:

2169-3536 © 2014 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

- Input 9800GX2 (GT4: 2353) measurement tool data.
- Use this data to parameterize the 9800GX2 (GT4: 2353) baseline model.
- Transform the baseline model to an NV280 (GT4: 2353) model by revising the baseline parameters and evaluate error.
- Project for NV280 performance using Amdahl's Law.

The remainder of this paper is organized as follows. Section II presents an overview of the GPU general architecture, Section III describes our method derivation, Section IV characterizes NV280, Section V introduces our performance analysis for NV280, Section VI presents related work, and conclude in Section VII.

II. GPU OVERVIEW

A Graphics Processing Unit (GPU) [2] is designed for highly paralleled operations as compared to a Central Processing Unit (CPU). A GPU has significantly faster and more advanced memory operations as compared to CPU. The Architecture of a GPU is based on a parallel array of many programmable processors. The CPU receives geometry information from the CPU and inputs and provides a picture frame as an output. The GPU unifies vertex geometry, pixel shader processing and parallel computing on the same processor. Some functions of GPU includes texture filtering, rasterization, raster operations, anti-aliasing, compression, decompression, display, video decoding, and high definition video processing. In general, the GPU consists of six pipeline stages described as follows:

1- *Input Assembler (IA):* The IA receives 3D representation of a scene as a collection of geometric parameters such as points, lines and vertices. The input data may also include collection of textures to be displayed. The IA stage gathers vertex data from up to eight input streams attached to vertex buffers and converts data items to a canonical format (e.g., float32). Vertex data is than fed into IA via a vertex cache used to cache and organize indexed primitive vertex data stream.

2- *Vertex Shaders (VS):* In general, VS determines how shadows and lighting interacts with surface to be rendered. VS are used to transform vertices from object to clip space by reading un-transformed vertex. They transform the vertex and write the resulting transformed vertex to output registers. VS can shade geometry in the lightning sense by calculating the lightning equation per vertex, but they can also manipulate all the vertex data. VS can manipulate properties such as texture coordinate and color. The output of the VS goes into the next stage called Clip stage.

3- *Clip (CC):* After vertex shader is transformed in either the fixed-function pipeline or a vertex shader, it is passed along to the clipper. Clipping is performed with the viewing frustum. Traditionally, primitives that lay partially or completely off-screen must be clipped to the screen or viewport boundary. In other words, in this stage, hardware make decisions about what geometry should be carried onto the next stage, and which piece of geometry does not need to be processed, it can throw away anything the viewer cannot see.

4- *Rasterization State (RS):* Handles clipping, culling, triangle setup and rasterization/multisampling. In general, RS is responsible for reading and writing depth and stencil, doing the depth and stencil comparisons, reading and writing color, and blending. RS transforms vectorized input into a bitmap form such as 2D array of pixels.

5- *Pixel Shader (PS):* PS computes the color and other attributes of each pixel. Usually the PS is driven by vertex shaders. PS allow for more flexibility with pixels, such as allowing interesting effects in the way that individual pixels are selected, blended or rendered. PS is used to perform per-pixel lightning on simple geometry.

6- *Output Merger (OM):* Takes a fragment from the PS and performs traditional stencil and depth testing operations as well as rendering target blending.

In general, to process one frame, the frame computation has to go through the GPU 6 pipelined stage as shown in Figure 1. In our model, we analyze performance bottlenecks in all these six pipeline stages.

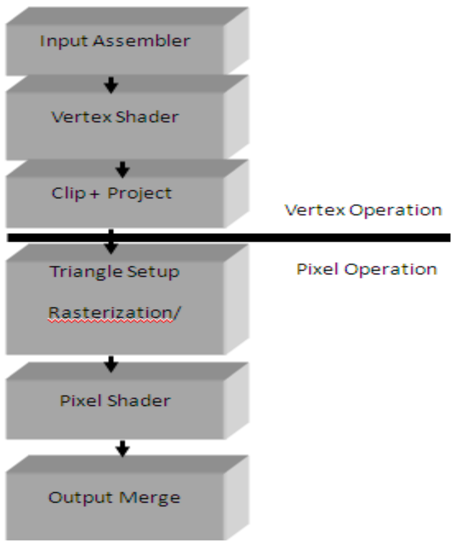

FIGURE 1. GPU pipeline block diagram.

Given the amount of computations required to process one frame at each pipeline stage, it is obvious that the GPU pipeline is full of performance bottlenecks waiting to happen.

III. METHOD DERIVATION

The focus of this development is the DX10 [8] architecture (9800GX2). The specific graphics workload used for this development is frame GT4: 2353. This means, we used frame # 2353 in GT4 3D benchmark. To analyze performance for a different frame, one needs to rerun the Nvidia NVtune [6], [7] performance measurement tool with that frame, then use that measured data as an input to the performance projection method.

A. BASELINE INPUT FILES

The Nvidia NV Tune utility provides the measurements used to drive the graphics model input data. The following is a set of eight input data sets required as an input to the model:

- Data98_fq – contains six pipeline-stage clock speeds (in MHz) for the six pipeline stages.
- Data98_alu – contains the number of parallel processors (in shaders which is the number of cores) at each pipeline stage for each draw call.
- Data98_t0 – contains the native time for a pipeline stage processor to process an element (in stage clock ticks/element/processor). Data98_t0 contains one value for each pipeline stage and draw call.
- Data98_bw – contains the effective bandwidth (bw) of a given pipeline stage running a given draw call at a given clock speed (in stage clock ticks/element/stage). There are three source files corresponding to three General Memory (GM) frequencies (500MHz, 750MHz, and 1000MHz).
- Data98_T – contains the measured values of the times (T_{meas}) for the 9800GX2 graphics pipeline to process draw calls (in nanoseconds). There are three source files for Data98_T corresponding to three GM frequencies.
- Data98_eta – contains η, the number of elements processed at each pipeline stage for each draw call (in elements/stage/draw call).
- Data98_shdr – contains indices designating shader programs running at the VS (Vertex Shader) and PS (Pixel Shader) pipeline stages for each draw call.
- Data280_T – contains the measured values of the times (T_{meas}) for the NV280 graphics pipeline to process draw calls (in nanoseconds).

The graphics model is configured to output the NV280-specific parameter changes to the baseline model, $T_{meas,0}$ and its projected value for each draw call as well as projection error which we will discuss in later sections.

B. BOTTLENECK-NB (BASELINE) AND T_{MIN}

According to the pipeline hypothesis, the simplest nontrivial estimate for T_{min} is

$$T_{\min}(c,f) \equiv \max_{1\le s\le 6} \{\eta(s,c)/bw(s,c,f)\} \tag{1}$$

where

$$\begin{aligned} T_{min} &= \text{The maximum of the six } \eta/bw \text{ ratios,}\\ s &= \text{Pipeline stage index } (\le 6, \text{ for a six stage pipeline}),\\ c &= \text{Draw call index (typically } \le 1200 \text{ per frame) and}\\ f &= \text{GM clock speed index } (\le 3). \end{aligned}$$

Each ratio is just the time for the corresponding stage to process η elements at its effective bandwidth (BW). We take each ratio to be the shortest possible processing time for the corresponding stage. Assuming that the pipeline stages process elements independently and in parallel, the minimum time for the graphics pipeline to process a draw call is the largest of these ratios. We expect that $T_{meas} > T_{min}$ and define $BN(c,f)$ to be the index of the bottleneck stage, i.e., the stage for which η/bw is the largest.

C. PROCESSING TIME AND NUMBER FRACTIONS

Table 1 gives an indication of the relative numbers (fr_num) of draw calls bottlenecked at various pipeline stages and the corresponding relative amounts of processing time (fr_tme) spent at these bottlenecks.

TABLE 1. NV GeForce 9800 measured time for each pipeline stage using 3DMarkVantage GT4 frame # 2353.

BN (Bottleneck Stages)	Draw calls / frame(%)	Processing Time
1- (Input Assembler -IA)	0.8	0.371
2- (Vertex Shader -VS)	0	0
3- (Clip - CC)	0.084	0.048
4- (Raster Operation -ROP)	0	0
5- (Pixel Shader - PS)	0.058	0.142
6- (Output Merger -OM)	0.022	0.308

From Table 1, 80% of the 1200 draw calls considered in GT4: frame # 2353 is bottleneck in IA stage. The time to process these draw calls is 37.1% of the total draw call processing time. IA has a large number of short draw calls. OM has a small number of large draw calls. No draw calls are bottlenecked on VS or ROP stages.

D. ERROR ANALYSIS

To determine the error for our analysis, we take error as a function of two timing parameters T_{min} and T_{meas} which refers to minimum and measured time respectively. Based on the data collected, T_{min} is not a good estimation for T_{meas}. To analyze the discrepancy, consider the relative estimation error Err defined by

$$Err(c,f) \equiv \frac{T_{meas}(c,f) - T_{\min}(c,f)}{T_{meas}(c,f) + T_{\min}(c,f)}. \tag{2}$$

Graphs of $Err(f,c)$ vs. $T_{min}(f,c)$ for various GM clock speeds (f) and bottleneck stages are shown in Figures 2–5. For example, Figure 2 shows points (Err, T_{min}), for which $f = 1$ (500 MHz GM clock speed), and b = $BN(c,f) = 1$ (IA bottleneck stage). The same method can be applied to other GM frequencies. By analyzing Figures 2–5, we notice the graphs are high on the left for small T_{min}. There is lots of variation at lower T_{min}, then they all slope off to the right for larger T_{min}. In other words, at lower T_{min} $Err(f,c)$ is high, and fluctuates before it starts to level off at higher T_{min}. This observation is similar in all other stages as shown in Figures 2–5. We conclude that T_{min} vs. $Err(f,c)$ presents

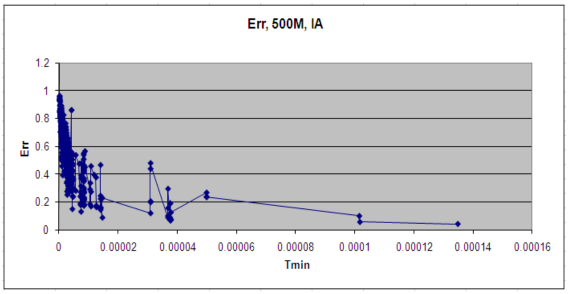

FIGURE 2. Error, GM Freq = 500 MHz, IA stage.

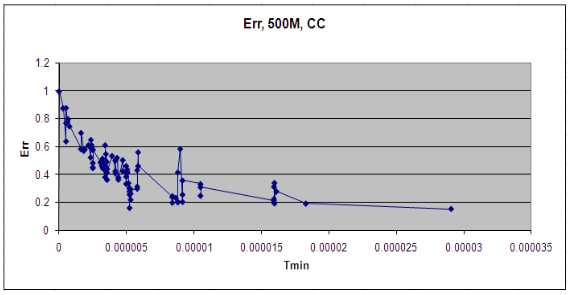

FIGURE 3. Error, GM Freq = 500 MHz, CC stage.

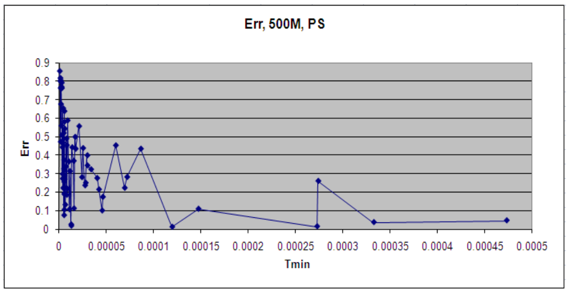

FIGURE 4. Error, GM Freq = 500 MHz, PS stage.

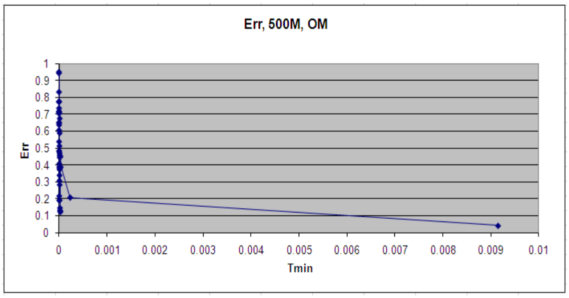

FIGURE 5. Error, GM Freq = 500 MHz, OM stage.

an exponential-like curve characteristic which is discussed in section 3.5 by introducing an error trend line.

E. E TREND LINE APPROXIMATION

It is hard and unnecessary to estimate the noise, instead we estimate its trend. To account for the noise, we fit the $Err(T_{min})$ graphs shown in the previous section with trend lines of the form:

$$Err\left(T_{\min}\right) = \exp\left(A - B \cdot T_{\min}\right), \tag{3}$$

where T_{min} is taken to be an independent variable, and parameters A and B vary from graph to graph, i.e., A and B depend on the GM clock speed and bottleneck stage. The exponential form follows the broad data trends. To find a best exponential fit for each $Err(T_{min})$ graph, we want values $A(b,f)$ and $B(b,f)$ for each bottleneck stage b and GM clock speed f that minimize the usual Linear Least Squares (LL2) function which derives the following equation:

$$LL2_{b,f}\left(A,B\right) \equiv \sum_{c:BN=b} wt\left(T_{\min}\left(c,f\right)\right) \times \left[\begin{array}{c}\ln\left(Err\left(c,f\right)\right)\\ -A\left(b,f\right)-B\left(b,f\right)\cdot T_{\min}\left(c,f\right)\end{array}\right]^2 \tag{4}$$

The summation is over draw calls with given GM clock speed f which also satisfy the constraint $BN(c,f) = b$, the given bottleneck stage. To derive LL2 estimation for A and B, we need first to fit a given set of points such that

$$\left\{\left(T_{\min}\left(c\right), Err\left(c\right)\right) \mid BN\left(c\right) = b\right\} \tag{5}$$

with a trend line of the form

$$E(T) \equiv \exp\left(A - B \cdot T\right) \tag{6}$$

where T is an independent variable and parameters A and B have yet to be determined.

We determine the best A and B by applying the conventional Linear Least Squares (LL2) criterion with possibly one small technical exception. Instead of trying to find A and B that minimize weighted sums of squares of differences $Err(c) - \exp\left(A - B \cdot T_{\min}(c)\right)$, we try to find A and B that minimize weighted sums of squares of differences $\ln\left(Err(c)\right) - \left(A - B \cdot T_{\min}(c)\right)$. Allowing this exception permits us to solve a linear rather than a nonlinear system of equations for A and B. The linear preference is obvious. Let $\{wt\ (c)\}_{BN(c)=b}$ denote a given set of weights for the data points. We simplify the notation by taking $e\left(c\right) = \ln\left(Err\left(c\right)\right)$ and $T(c) = T_{min}(c)$. Then we seek A and B that minimize the functional

$$LL2\left(A,B\right) \equiv \sum_{BN=b} wt\left(c\right)\left[e\left(c\right) - A + B \cdot T\left(c\right)\right]^2 \tag{7}$$

A necessary condition for LL2 to achieve a minimum at A and B is that A and B satisfy the equations

$$\frac{\partial LL2}{\partial A} = 0, \quad \text{and} \quad \frac{\partial LL2}{\partial B} = 0.$$

Therefore,

$$\frac{\partial LL2}{\partial A} = 0 = 2\sum_{BN=b} wt\left(c\right)\left[e\left(c\right) - A + B \cdot T\left(c\right)\right]\left(-1\right). \tag{8}$$

After rearranging,

$$\left(\sum_{BN=b} wt\left(c\right)\right)A - \left(\sum_{BN=b} wt\left(c\right)T\left(c\right)\right)B = \left(\sum_{BN=b} wt\left(c\right)e\left(c\right)\right). \tag{9}$$

A similar treatment of the $\partial LL2/\partial B = 0$ equation yields to

$$\left(\sum_{BN=b} wt\,(c)\,T\,(c)\right)A-\left(\sum_{BN=b} wt\,(c)T\,(c)^2\right)B$$
$$=\left(\sum_{BN=b} wt\,(c)\,e\,(c)\,T\,(c)\right). \tag{10}$$

To simplify the notation, take

$$u \equiv \sum_{BN=b} wt\,(c)\;T\,(c)^2,$$
$$v \equiv -\sum_{BN=b} wt\,(c)\;T\,(c),$$
$$w \equiv \sum_{BN=b} wt\,(c),$$
$$x \equiv \sum_{BN=b} wt\,(c)\;e\,(c),\quad \text{and}$$
$$y \equiv -\sum_{BN=b} wt\,(c)\,T\,(c)\;e\,(c).$$

The solution for A and B yields then to

$$A=\frac{ux-vy}{uw-v^2}, \tag{11}$$

and

$$B=\frac{wy-vx}{wu-v^2}. \tag{12}$$

We take $wt\,(T_{\min}) \equiv T_{\min}^{w}$, a special class of functions of T_{min} with free parameter w. We have focused on minimizing a weighted sum of terms of the form $[ln(Err) - ln(E)]^2$ rather than a weighted sums of terms of the form $[Err - E]^2$. As a result, we obtain A and B by solving a linear system of equations. Otherwise, the system would be nonlinear. $A(w, b, f)$ and $B(w, b, f)$ have been computed for 9800GX2 input data, various$b$ and f, and $w = 0.000$, 0.125, 0.250, 0.500, and 1.000. The results for the IA stage are shown in Figures 6-7. The same method can be used for the remaining pipeline stages.

In Figure 6 and 7, $LL2_{w,b,f}(A, B)$ achieves its minimum at the $A(w, b, f)$ and $B(w, b, f)$ estimates computed by the model. Let

$$E(w,b,f,T_{\min}) \equiv \exp\left(A\,(w,b,f) - B\,(w,b,f)\cdot T_{\min}\right) \tag{13}$$

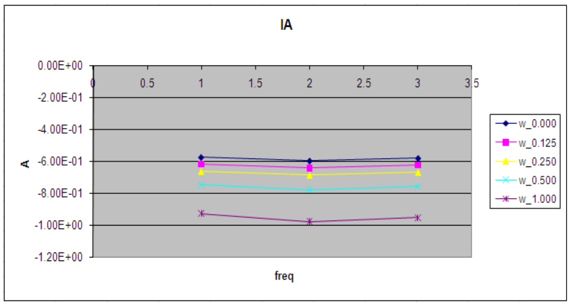

FIGURE 6. IA Stage – *A* coefficient variation with respect to 3 GM frequencies (500MHz, 750MHz, and 1000MHz) for 5 different W values.

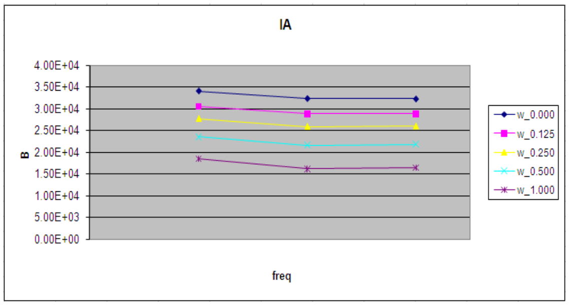

FIGURE 7. IA Stage – *B* coefficient variation with respect to 3 GM frequencies (500MHz, 750MHz, and 1000MHz) for 5 different W values.

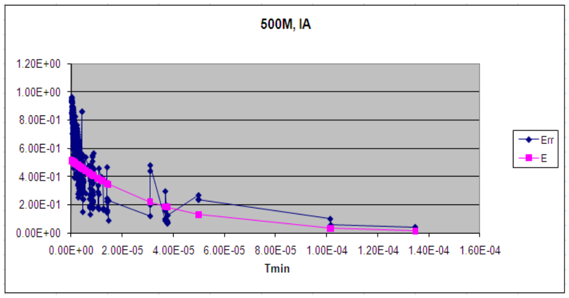

FIGURE 8. *Err* and *E* vs. T_{min} for IA stage, were *E* represents the exponential like curve.

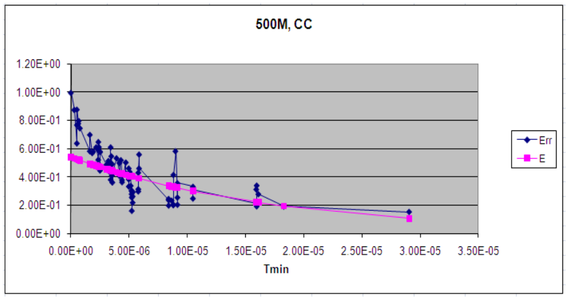

FIGURE 9. *Err* and *E* vs. T_{min} for CC stage, were *E* represents the exponential like curve.

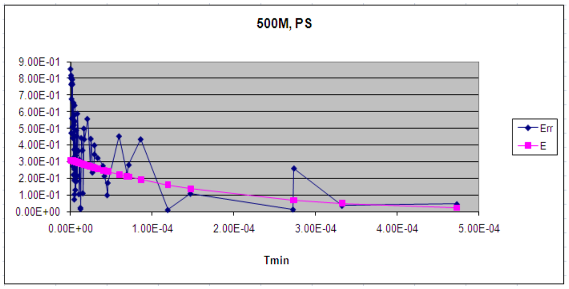

FIGURE 10. *Err* and *E* vs. T_{min} for PS stage, were *E* represents the exponential like curve.

denote the trend line through the set of points. Sample graphs of *Err* and *E* vs. T_{min} are shown in figures 8–11, for $w = 0.250$. We notice that the derived exponential-like curve E can be used as an estimate *Err*.

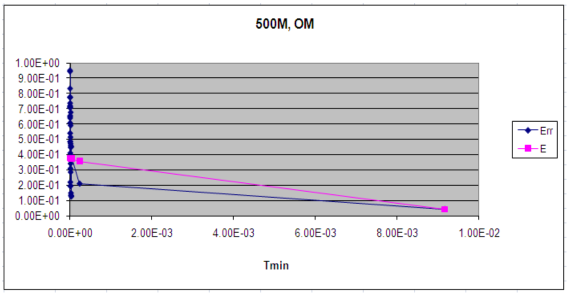

FIGURE 11. ***Err* and *E* vs. T_{min} for OM stage, were *E* represents the exponential like curve.**

F. T_{EST} FOR BASELINE

Recall that

$$Err\,(c,f) \equiv \frac{T_{meas}\,(c,f) - T_{\min}\,(c,f)}{T_{meas}\,(c,f) + T_{\min}\,(c,f)}. \tag{14}$$

The approximation

$$E\,(w,b,f,T_{\min}\,(c,f)) \;\sim\; Err\,(c,f) \tag{15}$$

induces the approximation

$$E\,(w,b,f,T_{\min}) \sim \frac{T_{est}\,(c,f) - T_{\min}\,(c,f)}{T_{est}\,(c,f) + T_{\min}\,(c,f)} \tag{16}$$

The discrepancy between T_{meas} and T_{est} is the fluctuation of Err about the trend line E. If these trend lines have been well chosen, then the fluctuations above the trend line largely cancel the fluctuations below the trend line. If we allow this cancellation in our draw call projection error estimates, then T_{est} should be a close estimate of T_{meas}. Solving for T_{est}, we obtain

$$T_{est}(w,c,f) = F\,(w,BN\,(c,f)\,,f,T_{\min}\,(c,f))\cdot T_{\min}\,(c,f) \tag{17}$$

where

$$F(w,b,f,T) \equiv \frac{1+E\,(w,b,f,T)}{1-E\,(w,b,f,T)}, \tag{18}$$

and

$$E(w,b,f,T) \equiv \exp\left(A\,(w,b,f) - B\,(w,b,f)\cdot T\right). \tag{19}$$

We have seen that T_{min} is a poor estimate for T_{meas}. The obstacle is Err (the noise). T_{est} is a revised estimate for T_{meas} that accounts for the trend line in Err but not for the fluctuations about this trend line.

G. ERROR ESTIMATE

To assess the T_{est} estimate, we consider the following definition of the draw call performance projection error:

$$\begin{aligned} &Err(w,f)\\ &\equiv \left|\sum_c \left(\frac{T_{meas}\,(c,f)}{\sum_k T_{meas}\,(k,f)}\right)\frac{T_{meas}\,(c,f) - T_{est}\,(w,c,f)}{T_{meas}\,(c,f)}\right|\\ &= \left|1 - \frac{\sum_c T_{est}\,(w,c,f)}{\sum_c T_{meas}\,(c,f)}\right| \end{aligned} \tag{20}$$

The error estimate for a draw call is proportional to the actual error. Given a preference for Err, we should also take $w \equiv 0.00$. The goal for this model is to analyze frame retirement time on a different graphics processor unit. The frame retirement time is proportional to $\sum_c T_{meas}\,(c,f)$, as an approximation. Therefore we would want to develop a model that produces T_{est} estimates of T_{meas} such that $\sum_c T_{est}\,(c,f)$ is close to $\sum_c T_{meas}\,(c,f)$, i.e., such that Err is small.

IV. CHARACTERISTICS OF NV280

We now consider a more demanding test in which we re-parameterize this baseline model with values characteristic of the NV280 platform as shown in Table 2. The remainder of this section describes this process and the derivation method to estimate the measured NV280 draw call processing times.

TABLE 2. Parameter changes for NV280.

Parameter	9800GX2	NV280
GM clock speed (MHz)	500, 750, 1000	900
VS processor count	16	240
PS processor count	128	120
GM bus width (bits)	384	512
IA clock speed (MHz)	600	512
VS clock speed (MHz)	1500	1180
CC clock speed (MHz)	600	512
ROP clock speed (MHz)	600	512
PS clock speed (MHz)	1500	1180
OM clock speed (MHz)	600	512

A. BANDWIDTH (BW) INTERPOLATION FOR NV280

Re-parameterizing the baseline model for NV280 takes three simple steps. The first is to estimate a value for bw for NV280 by interpolating measured baseline values. In particular, for each given draw call and pipeline stage, let bw_1, bw_2, and bw_3 denote the baseline effective bandwidths at the baseline GM clock speeds $fq_1 < fq_2 < fq_3$. Let bw_0 denote the effective bandwidth interpolated at fq_0, the NV280 GM clock speed.

If $fq_0 \leq fq_2$, then

$$bw_0 \equiv bw_2 + \left(\frac{fq_0 - fq_2}{fq_1 - fq_2}\right)(bw_1 - bw_2)\,. \tag{21}$$

Otherwise, take

$$bw_0 \equiv bw_2 + \left(\frac{fq_0 - fq_2}{fq_3 - fq_2}\right)(bw_3 - bw_2). \tag{22}$$

B. T_{MIN} AND BOTTLENECK (BN) FOR NV280

We use Equation (1) to estimate T_{min} and BN for both 9800GT2 and NV280. However, to use it for NV280, we first modify each η/bw ratio to account for parameter differences between 9800GT2 and NV280. In fact, the second step in re-parameterizing the baseline model is to account for these differences by the substitution

$$\frac{\eta(s,c)}{bw_0(s,c)} \leftarrow r_{proc}(s,c) \times \left[r_{mxu}\left(\frac{\eta(s,c)}{bw_0(s,c)} - \frac{t_0(s,c)}{n_{proc}(s,c)}\right) + r_{stg}(s)\left(\frac{t_0(s,c)}{n_{proc}(s,c)}\right)\right] \tag{23}$$

where

$$r_{proc}(s,c) \equiv \frac{number\ of\ 9800GX2\ stage\ s\ parallel\ processors}{number\ of\ NV280\ stage\ s\ parallel\ processors},$$

$$r_{mxu} \equiv \frac{9800GX2\ GM\ \ bus\ width}{NV280\ GM\ \ bus\ width},$$

and

$$r_{stg}(s) \equiv \frac{9800GX2\ stage\ s\ clock\ speed}{NV280\ stage\ s\ clock\ speed}$$

for stage s and draw call c.

This substitution can be easily understood by first writing the processing time $\eta(s,c)/bw_0(s,c)$ in the baseline context as the sum of two terms:

$$\frac{\eta(s,c)}{bw_0(s,c)} = \left(\frac{\eta(s,c)}{bw_0(s,c)} - \frac{t_0(s,c)}{n_{proc}(s,c)}\right) + \left(\frac{t_0(s,c)}{n_{proc}(s,c)}\right) \tag{24}$$

The first term is an estimate of the portion of this time that is somehow dependent on GM. To scale this estimate from 9800GX2 to NV280, we simply multiply by r_{mxu}. The second term is an estimate of the portion of this time that is independent of GM. To scale this estimate from 9800GX2 to NV280, we simply multiply by $r_{stg}(s)$. We multiply the sum of these two scaled terms by $r_{proc}(s,c)$ to account for the relative change in the number of parallel processors in going from 9800GX2 to NV280. (Increasing the number of processors decreases the processing time.)

Given these estimates for the η/bw ratios in the NV280 context, we estimate T_{min} and BN for NV280 just as we estimated T_{min} and BN for 9800GT2.

C. INTERPOLATING A AND B FOR NV280

The third step in re-parameterizing the baseline model for NV280 is to interpolate $A(w,b,f)$ and $B(w,b,f)$ at the NV280 GM clock speed. We proceed just as we did in section 4.1. The result is below.

If $fq_0 \le fq_2$, then take

$$A_0 \equiv A_2 + \left(\frac{fq_0 - fq_2}{fq_1 - fq_2}\right)(A_1 - A_2), \quad \text{and}$$
$$B_0 \equiv B_2 + \left(\frac{fq_0 - fq_2}{fq_1 - fq_2}\right)(B_1 - B_2). \tag{25}$$

Otherwise,

$$A_0 \equiv A_2 + \left(\frac{fq_0 - fq_2}{fq_3 - fq_2}\right)(A_3 - A_2), \quad \text{and}$$
$$B_0 \equiv B_2 + \left(\frac{fq_0 - fq_2}{fq_3 - fq_2}\right)(B_3 - B_2). \tag{26}$$

D. TEST FOR NV280

After re-parameterizing the baseline model for NV280, we apply this re-parameterized model to compute T_{est} for NV280. The result is

$$T_{est,0}(w,c) = F_0\left(w, BN_0(c), T_{\min,0}(c)\right) \cdot T_{\min,0}(c), \tag{27}$$

for c satisfying the constraint $BN(c) = b$, where

$$F_0(w,b,T) \equiv \frac{1 + E_0(w,b,T)}{1 - E_0(w,b,T)}, \tag{28}$$

and

$$E_0(w,b,T) \equiv \exp\left(A_0(w,b) - B_0(w,b) \cdot T\right). \tag{29}$$

V. PERFORMANCE ANALYSIS FOR NV280

The basics for Amdahl's law [1], [10] states that the performance improvement gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used. In other words, Amdahl's Law states that a system's overall performance increase is limited by the fraction of the system that cannot take advantage of the enhanced performance. Based on the Amdahl's Law regression method published in [13], we use the same method to analyze total frame retirement time for NV280 using the baseline established for NV280 as described in the previous sections. The pipeline model would generate two T_{est} data points for NV280 for two different GM frequencies relative to 9800GX2 baseline. The performance analytical model requires at least two measured data points to analyze performance for higher GM frequencies.

A. DERIVATION METHOD FOR PERFORMANCE ESTIMATION

The performance of a system can be divided into two distinct categories. The part which improves with the performance enhancement and is said to scale (variable a), and the part which does not improve due to the performance enhancement and is said to not scale or to be non-scaling (variable b). Note that a and b variables derived in this section are not related to A and B variables derived in previous sections. Based on the above definition, Amdahl's law can be written in the form of

$$T = T_0 + (T_1 - T_0)\frac{f_1}{f}, \tag{30}$$

where T_1 is the measured execution time at frequency f_1 and T_0 is the non-scale time. We can write T_0 in terms of a second measurement T_2 at f_2:

$$T_0 = \frac{T_2 f_2 - T_1 f_1}{f_2 - f_1}, \tag{31}$$

When we substitute Equation (30) for T_0, we obtain Amdahl's law in terms of two specific measurements without reference to T_0:

$$T = a + b\frac{1}{f}, \tag{32}$$

Where

$$a = \frac{f_2 T_2 - f_1 T_1}{f_2 - f_1}, \tag{33}$$

and

$$b = f_1 f_2 \left(\frac{T_1 - T_2}{f_2 - f_1} \right). \tag{34}$$

Variables a and b can be transformed to the score domain using $S \equiv 1/T$. For two data points, we will have (f_1, S_1) and (f_2, S_2), and for n data points, we will have $(f_1, S_1), \ldots, (f_n, S_n)$. We expect these points to satisfy an equation of the form (except for noise):

$$S_i \approx \frac{f_i}{a \cdot f_i + b}, \tag{35}$$

Due to noise, we cannot expect to find values for a and b that produce equality for each point i. In this case, we resort to the theory of linear least-squares estimation to obtain best estimates for a and b. In particular, given a and b, we take the error in our estimate for S_i in terms of f_i to be the difference between the measured and estimated value for S_i:

$$e_i \equiv \frac{1}{S_i} - \left(a + b \cdot \frac{1}{f_i} \right). \tag{36}$$

The best estimates for a and b are those that minimize the sum of the squares of these errors:

$$E = \sum_{i=1}^{n} e_i^2. \tag{37}$$

The estimates for a and b are those at which the values of the partial derivatives $\partial E/\partial a$ and $\partial E/\partial b$ are simultaneously zero. By computing these derivatives explicitly, we obtain equations satisfied by the best choices for a and b, which is the best functional fit to the measured data. If the data points are in the Time Domain $(f_1, T_1), \ldots, (f_n, T_n)$, then, we can use the relationship of score to time, and determine the best estimate for T in terms of f. Evaluating the equations we can derive a and b in terms of T and f for n points. We can also calculate a and b values to construct the linear score line in terms of $S1, S2, f1$ and $f2$ by substituting $S \equiv 1/T$ in the a and b equations. We use a and b values to define a score line as $y = ax + b$, were a is the scaling part, and b is the non-scaling part.

B. EXPERIMENTAL RESULTS FOR NV280 PERFORMANCE ANALYSIS

The process to estimate frame retirement time performance for a given benchmark entails three steps. The first step is to determine what to project, for example higher memory frequency. The second step involves establishing a measured baseline from which to project while keeping all other variable but one fixed. The model established two data points for NV280 based on GeForce 9800GX2 baseline. The third step is to analyze performance for the desired memory frequency using Amdahl's Law method. In Figure 12, we have two different data points calculated by the pipeline model for NV280.

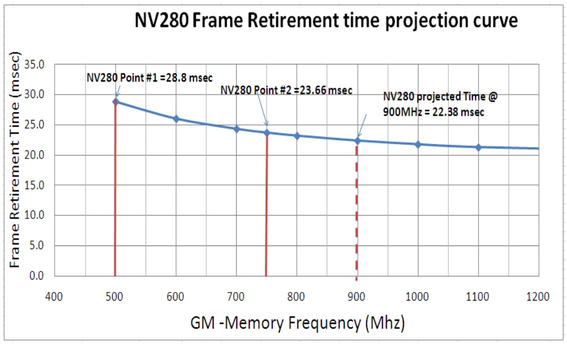

FIGURE 12. Performance analytical curve generated by amdahl's law method.

At data point#1, memory frequency is set at 500 MHz, the frame retirement time measured is 28.8msec. At data point#2, memory frequency is set at 750 MHz, the frame retirement time measured is 23.66msec. We will now use the projection line generated by Amdahl's law method to estimate the performance for NV280 at memory frequency set to 900 MHz which is 22.38msec as shown in Figure 12. For this experiment, the a and b values are the projection line intercept and slope values which are calculated as -111.73 and 0.5709 respectively. The linear score line is plotted in Figure 13 below. At $x = 0$ and $y = 0.5709$, it is the non-scaling part of the equation.

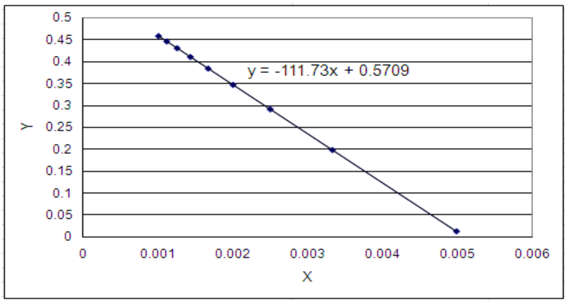

FIGURE 13. Linear performance line showing scaling and non-scaling factors in time domain generated by Amdahl's Law method.

The next step is to determine what is the maximum frame retirement time a GPU can achieve as GM memory frequency increases. The NVtune utility may not allow to set GM frequencies for higher values than the GPU can handle. In our method, we can analyze performance for these higher GM frequencies. Recall on the derivation steps for a and b values in section 5.1. We derived the equation $y = -111.73x + 0.5709$ at $x = 0$, and $y = 0.5709$. Now if we take the inverse of y, $1/y = 1/0.5709 = 1.751$. This result for $1/y$ concludes that the total frame retirement time for NV280 will never go below 17.51msec as memory frequency increases as shown in Figure 14. The *1/y* derived from scaling and non-scaling variables is set to be the lower bound for the time vs. memory frequency.

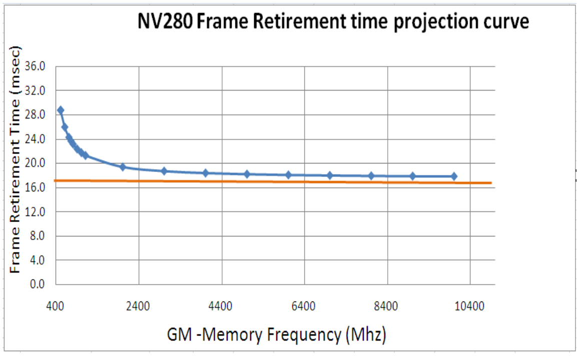

FIGURE 14. Lower bound @17.5msec as GM frequency increases.

The performance curve generated in Figure 14 using Amdahl's law method is the lower bound for frame retirement time on NV280 as GM memory frequency increases. NVTune utility may not allow setting GM frequencies to very high values due to hardware limitation, but Amdahl's law method enables the calculation of the maximum frame retirement time at these high GM frequencies. The analysis done on NV280 can be used to estimate frame retirement time for a different GPU of the same architecture as NV280, assuming that this GPU can achieve higher GM memory frequencies compared to NV280.

Scaling Efficiency (SE) is calculated using the following equation:

$$SE = \frac{y_2/y_1}{x_2/x_1} \tag{38}$$

were (x, y) is the projected frame retirement time curve data points.

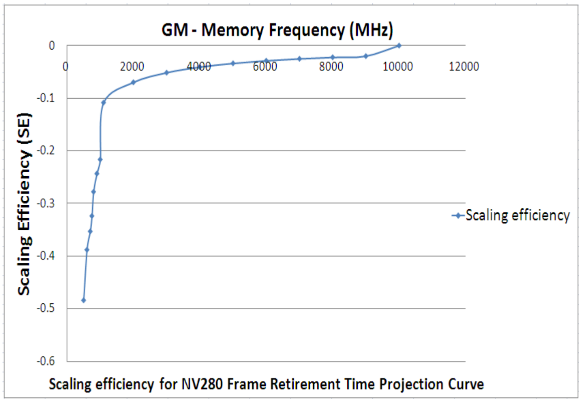

FIGURE 15. Scaling efficiency for NV280 frame retirement time projection curve.

By analyzing in Figure 15, the scaling efficiency curve for NV280 frame retirement time, we notice at lower GM memory frequencies, frame retirement time scales well with GM memory frequency. However at higher GM frequencies, the scaling factor seems to level off as we increase GM memory frequency. Using the Amdahl's law regression method described in this section and establishing the baseline for different Nvidia card discussed in previous sections, we test the model for 3 different Nvidia Cards as shown in Figure 16.

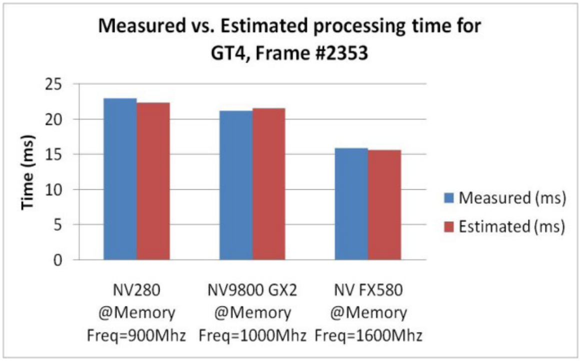

FIGURE 16. Measured vs. estimated frame retirement time for different Nvidia cards.

The model which consists of two parts, the first part establishing a baseline for a given graphics card to generate two data points, and the second part consists of using Amdahl's law to predict performance at different memory frequency. We then measure the actual card at that frequency to show the error variation between the model output and the measured data. For all three graphics cards shown in Figure 16, the error margin is $<5\%$ for all tested graphics cards.

VI. RELATED WORK

Several researchers have worked on analyzing GPU performance on given benchmarks using different analytical and simulated methods. In this paper, we presented an analytical method based on a mathematical derivation to calculate and analyze frame retirement time for a given GPU architecture on a specific frame. This method is an analytical method rather than simulation based method.

Moya, V. [14], implemented a performance analysis of shader processing for a given GPU using simulated and comparison methods.

Sibai, F. [15], presented performance characterization for 3Dmark benchmark focusing on multi-CPU, multi-GPU platforms. The paper focus on 3DMark characterization, and the dependence of 3D graphics on memory. The characterization method does not provide insight on the pipeline activities for GPU while rendering a specific frame from 3Dmark workload.

Liu, W. [16], identifies factors that determine performance of GPU-based applications and then classify them into different categories. The data characteristics are used to propose a performance model for each factor. The models were then used to estimate the performance of bio-sequence database scanning application on GPU.

VII. CONCLUDING REMARKS

In this paper, we presented an analytical method to analyze and project frame retirement time using Amdahl's law after establishing a measured baseline. Using the same method, we also determined the maximum frame retirement time a graphics card can achieve on a given benchmark for a specific frame. More importantly, the method is flexible as it can be applied to project performance for a different graphics card of same architecture at higher memory frequencies. The method does not require binary or simulations; it does however require a code to implement the performance analysis method. This method is limited because it can only project performance for graphics processors of same architecture specific to a certain frame. The same method can be implemented to analyze performance for GPU of different architectures; however, this will result in a projection curve shift as well as change in slope characteristics. Future work can be done by extending this model to analyze performance for graphics workloads or entire 3D benchmarks rather than individual frames. In our previous work [13] Amdahl's law method was used to analyze the sensitivity and performance curve for different memory frequency, core GPU frequency and different number of cores on a given GPU using different 3D benchmarks. The performance projection method we have is limited as we can only sweep one variable at a time, for example, in this paper, we changed GM frequency, in [13] we changed memory frequency, core GPU frequency, and number of cores. Future work on performance analysis method can implement multi-dimensional plots to analyze the performance (Frame Retirement Time, Frame per Second or Score) sensitivity curve using different frequency axis such as memory and core frequencies as well as different number of cores on a given benchmark.

REFERENCES

[1] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. New York, NY, USA: Wiley, 1991.
[2] GPGPU. (2005). *General-Purpose Computation Using Graphics Hardware*. [Online]. Available: http://www.gpgpu.org/
[3] K. Hoste, L. Eeckhout, and H. Blockeel, "Analyzing commercial processor performance numbers for predicting performance of applications of interest," to be published.
[4] *NV GeForce 9800 Spec*. [Online]. Available: http://www.nvidia.com/object/product_geforce_9800_gx2_us.html
[5] *Nvidia NV280 Spec*. [Online]. Available: http://www.nvidia.com/object/product_geforce_gtx_280_us.html
[6] *Nvidia PerfHUD Utility*. [Online]. Available: http://developer.nvidia.com/object/nvperfhud_home.html
[7] *Nvidia nTune Utility*. [Online]. Available: http://www.nvidia.com/object/ntune_2.00.23.html
[8] Microsoft Corp. *DirectX11*. [Online]. Available: http://www.microsoft.com/directx
[9] R. H. Saavedra and A. J. Smith, "Analysis of benchmark characteristics and benchmark performance prediction," *ACM Trans. Comput. Syst.*, vol. 14, no. 4, pp. 344–384, Nov. 1996.
[10] S. Krishnaprasad, "Uses and abuses of Amdahl's law," *J. Comput. Sci. Colleges*, vol. 17, no. 2, pp. 288–293, Dec. 2001.
[11] M. Annaratone, "MPPs, Amdahl's law, and comparing computers," in *Proc. 4th Symp. Frontiers Massively Parallel Comput.*, Oct. 1992, pp. 465–470.
[12] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, Jul. 2008.
[13] J. Issa and S. Figueira, "Graphics performance analysis using Amdahl's law," in *Proc. SPECTS Conf.*, Jul. 2010, pp. 127–132.
[14] V. Moya, C. Gonzalez, J. Roca, A. Fern, and R. Espasa, "Shader performance analysis on a modern GPU architecture," in *Proc. 38th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Nov. 2005, pp. 354–364.
[15] F. N. Sibai, "3D graphics performance scaling and workload decomposition and analysis," in *Proc. 6th IEEE/ACIS Int. Conf. Comput. Inf. Sci. (ICIS)*, Jul. 2007, pp. 604–609.
[16] W. Liu, W. Müller-Wittig, and B. Schmidt, "Performance predictions for general-purpose computation on GPUs," in *Proc. Int. Conf. Parallel Process. (ICPP)*, Sep. 2007, p. 50.
[17] *3D Mark Vantage*. [Online]. Available: http://www.futuremark.com/benchmarks/3dmark-vantage

JOSEPH A. ISSA received the B.E. degree in computer engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 1997, the M.S. degree in electrical engineering from San Jose State University, San Jose, CA, USA, in 2000, and the Ph.D. degree in computer engineering from Santa Clara University, Santa Clara, CA, USA, in 2012. Since 2013, he has been an Assistant Professor with the Department of Computer and Electrical Engineering, Notre Dame University, Zouk Mosbeh, Lebanon. His research interests include processor architecture and performance modeling.