Workload-aware Power Gating Design and Run-time Management for Massively Parallel GPGPUs

Kapil Dev, Sherief Reda
Brown University
{kapil_dev, sherief_reda}@brown.edu

Indrani Paul, Wei Huang, Wayne Burleson
Advanced Micro Devices, Inc.
{Indrani.Paul, WeiN.Huang, Wayne.Burleson}@amd.com

***Abstract*—Power gating (PG) is an effective power efficiency improvement technique. Future general-purpose graphics processing units (GPGPUs) will likely feature hundreds of compute units (CUs) and be power constrained, which leads to serious challenges to existing PG methodologies. In this paper, we propose novel design-time and run-time techniques to effectively implement power gating in future GPGPUs. Based on industrial models/measurement facilities, we show that designers must consider run-time parallelism within potential applications while implementing power gating designs to avoid incurring unnecessary design overheads. By scaling measurements from a real 28nm GPGPU to a hypothetical future 10nm node, we show that a PG granularity of 16 CU/cluster achieves 99% peak runtime performance without the excessive 53% design-time area overhead of per-CU PG. We also demonstrate that a run-time power management algorithm that is aware of the PG granularity leads to up to 18% additional performance through frequency-boosting under thermal-design power (TDP) constraints.**

***Index Terms*—GPGPUs, power gating, power management.**

I. Introduction

General-purpose graphics processing units (GPGPUs) are now commonly used in many high-performance computing (HPC) systems due to their high-performance and power efficiency (perf/W) [1]. Future petascale and exascale systems are likely to incorporate GPGPUs with hundreds of compute units (CUs) [2]. Emerging trends show that these CUs have to operate under tight power budgets to sustain reliable operating temperatures and avoid excessive leakage power or thermal runaway. As a result, not all CUs may always be powered on simultaneously at their maximum frequency [3]. Thus, it is necessary to dynamically adjust the number of active CUs through power-gating (PG) mechanisms based on the run-time needs of applications. However, PG introduces serious design and area overheads, which if applied liberally can negate its benefits. There is a tradeoff between design overheads and run-time performance and power efficiency.

This paper develops an integrated approach towards addressing PG challenges in potential future GPGPUs by analyzing: 1) design-time decisions where the benefits of fine-grain PG must be balanced against its overheads, and 2) run-time decisions where PG and frequency boosting need to be applied adaptively to control the number of active CUs based on the GPGPU design, the application needs, and the power budget. Specifically, the contributions of this paper are as follows.

- We demonstrate the need for an integrated solution to manage leakage power by incorporating *workload/run-time awareness* into the PG design methodology that determines the optimal PG granularity, and *design-awareness* into the run-time power management algorithm that finds the optimal number of CU to power gate.
- Using industrial scaling models and real hardware measurements, we project run-time parallelism trends of HPC applications to a future massively parallel GPGPU. We use these trends along with accurate PG area models to determine the optimal design choices for PG granularity (i.e., cluster size) that improve power efficiency with minimal run-time and design overheads.
- We propose a run-time power management algorithm that utilizes PG design knowledge to shift power from unused CUs to boost the frequency of active CUs, thereby leading to better performance and power efficiency.
- Compared to per-CU PG, we demonstrate that a runtime-aware design of a 16CU per cluster achieves 99% peak runtime performance without the excessive 53% design area overhead on a hypothetical GPGPU with 192CUs in 10nm. Further, we show that a run-time algorithm that is aware of the PG granularity leads to up to 18% higher performance under thermal-design power (TDP) limits.

The rest of the paper provides the related work, motivation, the proposed methodology, experimental results and conclusions of this work in successive sections.

II. Related Work

GPGPUs are being used to improve performance and energy efficiency of many classes of HPC applications [1]. Dynamic voltage and frequency scaling (DVFS), clock-gating, and PG are common techniques used to manage power and energy in multi-core CPU and GPGPU processors [4], [5], [6]. Further, based on application needs, run-time hardware reconfigurations are also used to improve power efficiency of both CPUs and GPUs [7], [8], [9], [10].

While Lee *et al.* analyzed throughput improvement of multi-core processors by power-gating and DVFS techniques [11], Majeed *et al.* proposed a PG-aware warp scheduler to improve the benefits of GPGPU power gating. The existing techniques [12], [13], [14], [15] are useful to improve the power efficiency of GPGPUs with underutilized resources or improve performance under TDP constraints. However, most of the previous studies investigated PG opportunities statically by assuming the finest level of power gating at per-CU/core level without considering area overhead or they did not consider the impact of design-time choices on the run-time performances.

In contrast to previous works that considered few CUs [12], [14], [15], our methodologies are geared for hundreds of CUs that will be available at future technologies.

III. Motivation & Goals

Future HPC systems are predicted to operate massively parallel GPGPUs under tight power and thermal constraints, which poses unique challenges as follows.

Leakage power in future massively parallel GPGPUs. The large number of CUs in potential future GPGPUs will result in high power consumption and power densities, and hence, higher temperatures across the chip. Although FinFET technology significantly reduces leakage power, it is predicted that the sub-threshold leakage ceiling for FinFET will be comparable to planar bulk CMOS due to self-heating [16]. Hence, leakage power can still be the dominant if all CUs are left powered on at high operating temperatures. High leakage causes frequency throttling for a tight power budget.

Workloads demonstrate diverse scaling trends. Fig. 1 shows the performance of three example HPC application kernels, `GEMM.sgemmNN`, `LULESH.CalcHourGlass`, and `XSBench.calcSrtd`, as a function of the number of active CUs (Section IV describes the used methodology). The performance of each kernel is normalized to its own minimum baseline. We observe that HPC applications present various parallelism needs due to their diverse compute and memory behavior, requiring a run-time power management unit that could adaptively control the number of CUs by power gating unused CUs. For example, `XSBench.calcSrtd` has a peak performance at 124 CUs, beyond which significant cache thrashing occurs and performance degrades.

PG granularity is critical to performance and power management. Power savings from power gating can be used to boost frequencies of the remaining active CUs for higher performance under a given power budget. The amount of savings depends on the PG granularity–defined as the minimum number of CUs that can be power gated at once usually a design-time decision. A finer PG granularity results in large design area overheads, whereas an overly coarse granularity results in excessive leakage power and run-time performance degradation, especially for applications that use CUs with non-exact multiples of the PG granularity. Often, the existing run-time power management systems have no knowledge of the underlying PG design leading to sub-optimal decisions.

In summary, PG techniques that do not consider run-time opportunities during design and design choices during run-time can lead to poor performance and power efficiency with large design overheads. So, our **goal** is to couple the design-time PG granularity with the run-time opportunities to maximize performance and power efficiency of future GPGPUs.

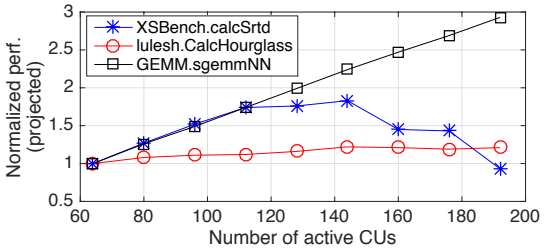

Fig. 1. Performance scaling of three example kernels.

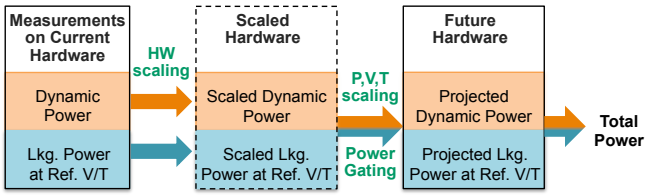

Fig. 2. Power scaling methodology.

IV. Proposed Methodology

The proposed methodology for power gating of future GPGPUs involves three components. First, we propose in Subsection IV-A a methodology to scale power and performance measurements on existing GPGPU devices to future devices with similar micro-architecture. Using these extrapolated measurements, we propose in Subsection IV-B a methodology to analyze the impact of different PG granularity choices on leakage power saving with respect to the available parallelism in applications. Third, we propose a run-time power management technique in Subsection IV-C that utilizes workloads' characteristics and the PG granularity to maximize performance and power efficiency under a fixed TDP.

A. Performance and Power Scaling

We use a 3-step methodology, as proposed by Dev *et al.* [17] to estimate performance and power for a potential future GPU architecture with 192 CUs at hypothetical 10 nm technology. Fig. 2 shows the overall methodology for power estimation. First, we take real measurements of performance and power (dynamic and leakage) for all kernels/applications at different CU-count, CU-frequencies, and memory-bandwidths (BW) on an existing GPU device (in 28nm) described in Section V.

Second, we extrapolate measurements on the baseline device to potential future devices using the same compute-to-memory ratio for both devices. We consider a compute throughput proportional to the product of number of CUs and CU frequency, and a memory BW proportional to memory frequency. We use the idea that performance scales proportional to the scaling of the compute throughput and BW of a GPU, given the same compute-to-memory ratio.

Finally, we apply the effect of changing the process technology (including the introduction of FinFET from 14nm), voltage, frequency, and temperature on both dynamic and leakage power to project the total processor power based on models from fabrication vendors. Our proposed PG methodology is quite generic and valid even if other scaling methods (e.g., detailed simulation based) are used.

B. Workload-Aware Design-time Analysis

Leakage power savings, and thereby potential frequency-boosting, from power gating unused CUs are workload-dependent. The finer the PG granularity, the higher the boosting potential. Also, beyond a certain granularity, if

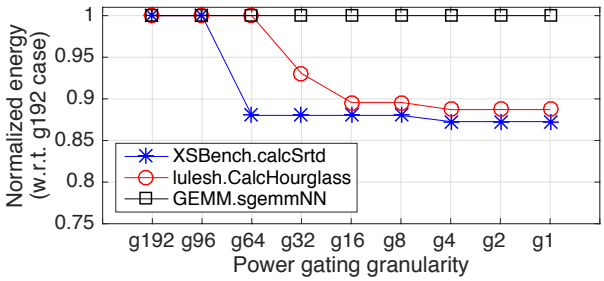

Fig. 3. Normalized energy of selected kernels at different PG granularities (g192: no power gating, g1: per-CU power gating).

the maximum current and frequency allowed for a given PG transistor-sizing are reached, further frequency boosting becomes impossible. Thus, we propose a methodology to evaluate leakage power, frequency boosting factor, and area overhead at different PG granularities as a function of the optimal number of CUs needed by an application at run-time.

Leakage power analysis. We model the effects of PG granularity on total leakage power by considering the optimal number of CUs needed by an application. We assume that there are N CUs in a GPGPU device and that the device has a PG granularity of s. If an application needs n CUs for its optimal performance, and if we denote the average leakage power of one CU by p_1, then the total leakage power, denoted by $P_L\,(n, s)$, of the system for n active CUs is given by:

$$P_L\,(n,s) = p_1 s \left\lceil \frac{n}{s} \right\rceil, \quad n \leq N \quad \text{and} \quad s \leq N, \tag{1}$$

where $\lceil \cdot \rceil$ denotes the ceiling operator. We note that for fixed number of active CUs, the leakage will be different for different PG granularities. For example, Fig. 3 shows the normalized energy versus PG granularity (*g192* with no PG to *g1* with per-CU PG) for 3 example kernels obtained through our projection models. We clearly see the "knee points", in terms of energy vs. granularity, across different kernels. We see that the energy reduction of these kernels almost flattens out after a granularity of 16. Thus, we seek an optimal PG design that balances the benefits of fine-grain granularity (performance and power efficiency) with the design cost (die area overhead).

Frequency boosting. The power saved from gating unused CUs can be used to boost the frequency and thereby the performance of certain kernels. The amount of boosting depends on the PG granularity of the design, the TDP, and the maximum die temperature of the device. For our analysis, we use the worst-case die temperature to ensure deterministic performance under a fixed cooling solution irrespective of the process and ambient variation across parts. Further, we observe that an increase in operating frequency will lead to an increase in switching electrical current, which requires a proportional increase in the size of the PG transistors to reduce the IR drop across them. The kernel with the maximum frequency boost dictates the size of the transistors.

PG area overhead. Implementing PG requires adding power gates, buffers, clamp cells, I/O buffers, and control logic to the design [18], which increases its total area. Further, the area overhead depends on the granularity at which the PG

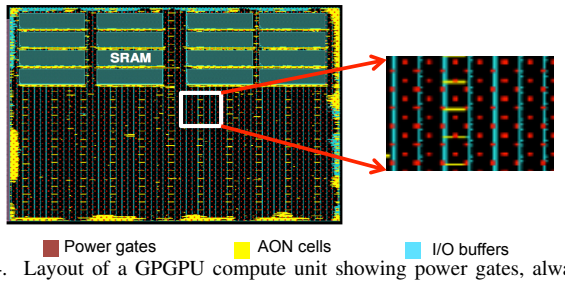

Fig. 4. Layout of a GPGPU compute unit showing power gates, always-on (AON) cells and I/O buffers [18]. The snapshot on the right shows the zoomed area marked by the white rectangle.

is implemented in the design and the amount of frequency boosting that can be allowed under TDP during run-time. The area overhead, A_{ov}, due to PG can be expressed as

$$A_{ov} = A_{gates} + A_{aon} + A_{cntrl}, \tag{2}$$

where A_{gates}, A_{aon} and A_{cntrl} denote the area overheads due to power gates, always-on (AON) cells, and control logic, respectively. A_{gates} scales with the frequency-boost, which depends on both the PG granularity and kernels. That is

$$A_{gates} = A_{gates\_f_0} \frac{I_{fboost\_max}}{I_{f_0}}, \tag{3}$$

where $A_{gates\_f_0}$ is the area overhead of power gates at the nominal frequency, f_0, and I_{fboost_max} denotes the current at the maximum frequency-boosting factor, which is decided by the kernel with the largest power slack with respect to TDP.

As the PG cluster-size, s, increases, fewer number of AON cells are needed due to reduced intra-cluster signals. Thus, A_{aon} is modeled as a product of the perimeter of the cluster and the number of PG clusters in the device; for a cluster size s, where its CUs arranged as $s_v \times s_h$ grid, the perimeter of the cluster will be $2\,(s_v + s_h)$. For N number of total CUs,

$$A_{aon} \propto (s_v + s_h)\,\frac{N}{s}, \quad \text{such that} \quad s = s_v \times s_h, \tag{4}$$

Finally, the area overhead due to PG control logic and associated sleep/wake signals is modeled as a linear function of the number of PG clusters; that is,

$$A_{cntrl} \propto \frac{N}{s}. \tag{5}$$

Fig. 4 shows the layout of a compute unit from an industrial GPGPU design that has power gates inserted in a checker board pattern [18]. The snapshot on the right shows different power gating components (i.e., power gates, always ON (AON) cells, and I/O buffers) in the zoomed area. We use this layout to compute the area overheads from the components used for implementing power gating in the design. The area overhead results are presented in Section V.

C. Design & Workload-Aware Run-time Management

We develop a simple and practical predictor that can estimate the number of required CUs and that can be implemented efficiently in a run-time algorithm with minimal hardware overhead and complexity, and maximize performance.

Run-time performance correlation. We studied the correlation between 20+ hardware performance counters with the performance for all kernels at different CU counts on our current hardware described in Section V, and we found that GPU compute utilization, namely, *VALUBusy*, has a very strong

Algorithm 1: Gradient-based algorithm to find optimal CU count and frequency for a kernel during run-time.

```
   Data: PG granularity (s), minimum step-size (Δn_0), nominal
         frequency (f_0)
   Result: Optimal CU count and frequency
1  Initiatlization(); k = 2;
2  // First find optimal CU count at nominal frequency (f_0)
3  // VALUBusy denoted as V; Gradient as G
4  while (ΔV > tol) OR (G <= 0) do
5      k = k + 1;
6      n_k=PredictNumCU(n_{k-1}, Δn_0, G);
7      P_k = PredictPower(n_k, f_0, s);
8      if P_k <= TDP then
9          Run kernel at n_k and measure V_k;
10         ΔV = (V_k − V_{k-1})/V_{k-1};
11         G = (V_k − V_{k-1})/(n_k − n_{k-1});
12     else
13         k = k − 1; // TDP exceeded
14         ReduceNumCU();
15     end
16 end
17 // Then find optimal operating frequency
18 while P_k < TDP do
19     Boost frequency; // Use PG granularity while varying frequency
20     PG_Granularity_PredictNumCU();
21 end
22 Optimal CU count = n_k at the highest V_k;
```

correlation (>0.99) with the performance of an application across all kernels. *VALUBusy*, also denoted as V, represents the percentage of GPU-time when vector instructions are processed; higher *V* means higher CU utilization and performance.

Power management algorithm. We propose a run-time power management algorithm that searches for the optimal CU count dynamically using application characteristics and PG granularity information. The algorithm applies a gradient-based analysis for power gating idle CUs and adjusting the frequency of active CUs for an application kernel under a given TDP. This algorithm can be invoked at any reasonable sampling interval (per-kernel, application, or at fixed intervals). Shown in Algorithm 1, it has three main components: 1) initialization, 2) gradient computation, 3) configuration prediction.

1. *Initialization.* During initialization, we run the kernel at two different CU configurations and collect $VALUBusy$ (line 1 of Algorithm 1). The convergence time of our iterative algorithm depends on the difference in performance between these initial points and is a function of the actual optimal CU count of the kernel. The larger the difference, more iterations may be needed to converge.

2. *Gradient computation.* Next, we compute the gradient (G) for *VALUBusy* by taking the ratio of the change in the value of *VALUBusy* counter to the change in the active CUs (line 11 of the Algorithm). We denote the minimum step-size for change in CU count in the system as Δn_0. The algorithm predicts the number of active CUs for the current iteration (k) of the kernel from the number of active CUs and the gradient of *VALUBusy* counter with respect to the number of CUs in the previous step (line 6 of the Algorithm). That is

$$n_k = n_{k-1} + \Delta n_0 * \text{sign}\,(G) * \{1 + \text{round}\,(C * |G|)\}\,,$$

where, the parameter C controls the convergence rate. Due to adaptive and quantized step-sizes, it is possible to overshoot the maxima point in our proposed algorithm. Therefore, unlike standard gradient-ascent method, we reduce the number of CU count when the gradient is negative; we add 1 to avoid the algorithm from getting stuck at a fixed CU count when round($C*|G|$) = 0. The algorithm runs until the relative change in the optimization function (*VALUBusy*) is smaller than a specified tolerance value (tol). We set tol to 2%.

3. *PG design-aware configuration prediction.* To simplify the performance/power scaling models, either CU-count or operating frequency is changed at a time, and not both of them at the same time. With all these, we predict the optimal CU configuration based on the following cases:

Case 1: Activating optimal number of CUs under TDP at nominal frequency. Once the run-time algorithm predicts the optimal CU count for a kernel, it also predicts the corresponding power consumption based on the previous power readings using PG granularity size (line 7 of the Algo). If the predicted power consumption at the optimal CU configuration is more than the TDP, the algorithm reduces the CU count until the power constraint is met. In this paper, we focus on the overall PG methodology, so, some of the functions [e.g., PredictPower()] are not discussed in detail for brevity.

Case 2: Adjusting CU count and frequency to utilize power headroom. We boost frequency if there is power slack available after selecting the optimal number of CUs. The amount of boost is derived by computing gradients for *VALUBusy* with respect to frequency (similar to the line 11 of Algo). To this end, the algorithm varies the number of CUs within immediate integer multiples of the PG granularity and adjusts the frequency to meet the TDP (line 20). We choose CU scaling first because for HPC type parallel applications CU scaling provides better power efficiency than frequency scaling.

V. Experimental Results

Our experimental setup consists of an existing high-performance GPU with 32 CUs, each has one scalar unit and four 16-wide SIMD vector units. Without loss of generality, we evaluate the performance and power efficiency of a potential future massively parallel GPGPU architecture with 192 CUs (6× higher) at hypothetical 10 nm technology, mainly motivated by the performance and power efficiency demands projected for future exascale systems [2].

We first take hardware measurements across 25 kernels in 13 applications from the exascale proxy applications [19] and the Rodinia benchmark suite [20] on the baseline GPU by sweeping number of CUs (4 to 32 in steps of 4 CUs), CU frequency (300 MHz to 1 GHz in steps of 100 MHz), and memory interface frequency (475 MHz to 1375 MHz in steps of 150 MHz). This results in performance and power measurements (both dynamic and leakage power using internal registers) over 448 distinct hardware configurations. The measurements are scaled to future potential GPGPU using the methodology described in section IV. Further, we use the AMD CodeXL profiler to collect kernel-level performance counters on the existing GPU.

TABLE I
OPTIMAL NUMBER OF CUS FOR STUDIED KERNELS.

Kernel Name	Opt. #CUs	Kernel Name	Opt. #CUs
CoMD.EAM.advPos (K1)	124	GEMM.sgemmNN (K14)	192
CoMD.EAM.advVel (K2)	104	GEMM.sgemmNT (K15)	104
miniFE.dotprod (K3)	64	XSBench.bitSort (K16)	124
miniFE.waxpby (K4)	192	XSBench.calcSrtd (K17)	124
minife.matvec (K5)	184	XSBench.uGridSrch (K18)	152
MaxFlops.peak (K6)	192	bprop.adjW (K19)	104
lulesh.cacFBHour (K7)	184	bprop.lfwd (K20)	192
lulesh.integStress (K8)	184	cfd (K21)	184
lulesh.calcHourglass (K9)	140	hotspot (K22)	88
graph500.locRedNext (K10)	192	Device memory (K23)	64
graph500.botUpStep (K11)	124	Nbody (K24)	184
graph500.unionClear (K12)	64	kmeans.swap (K25)	80
kmeans.kernel_c (K13)	184		

Methodology validation. We validate the projection methodology by using measurements at configurations with low CU count to predict the performance and power at configurations with maximum 32 CUs on the current hardware device. Note that this does not involve technology scaling. The average errors in predicted execution time and power against the actual measurements for the studied kernels are 3.1% and 1.3%.

Further, we also validate the optimal CU predictor by comparing run-time algorithm predictions with oracle CU counts directly derived from off-line execution time estimates. We find that tracking *VALUBusy* leads to very accurate optimal CU count prediction. Table I gives the optimal CU count for all kernels (K1 to K25) studied in this paper. In contrast with the existing search algorithms, the proposed Algorithm 1 considers PG granularity and converges faster, in less than 6 iterations across all the kernels, (by avoiding disabling one CU at a time) in a massively parallel architecture. Since kernels usually go through tens of iterations, and dynamically changing hardware configurations requires only a few microseconds, the proposed algorithm introduces very little runtime overhead.

Evaluation results. With 150 W TDP constraint and the nominal 1 GHz frequency, Fig. 5.a,b show the execution time and energy for the selected kernels at different PG granularities using our design-aware run-time management algorithm, normalized to the baseline case where all CUs power gated together (i.e., g192). The lower energy in Fig. 5.b means higher power efficiency. Similarly, lower execution time means higher performance. Due to space constraints, we only show results from selected kernels for seven PG granularities that divide the total 192 CUs: 1, 8, 16, 32, 64, 96, 192 *CUs per cluster* that can be gated at the same time. This corresponds to 192, 24, 12, 6, 3, 2, and 1 independent PG domains.

Observation #1: *There is an optimal PG granularity shared by most workloads.* For most kernels, we see diminishing return beyond g16. *The improvement arises because kernels are TDP limited and the more leakage power saved from finer-grained PG can be leveraged to activate more CUs, which in general improves performance and power efficiency.* In fact, the largest performance improvement of 14% is seen between one PG domain (g192) and two PG domains (g96) for the entire GPU. However, beyond a certain PG granularity, the additional leakage saving becomes smaller, and hence the diminishing returns in performance and power efficiency with more silicon area overhead. Specifically, there is no significant performance and power efficiency benefit between g16 and g1. Two exceptions are the performance of `CoMD.EAM.advPos` (K1) and `hotspot` (K22), where we see no performance changes across different PG granularities as compared to the baseline because they never exceed TDP even without power gating and can run at their optimal CUs. Across 25 kernels we have investigated, we find that 16 CU cluster size (g16) is the finest granularity that is necessary. At g16, we find an average performance and power efficiency improvement of 21% and 30%, respectively, compared to the baseline case g192.

Observation #2: *Per-CU PG is an overkill.* We have seen that finer PG granularity provides better performance and power efficiency. However, the improvements come at the cost of design and area overhead. Although the area overhead due to control logic and AON cells keeps increasing with finer granularity, the area overhead due to power gates starts saturating (e.g., at g64 in Fig. 5.c) as the maximum frequency is attained. By comparing the performance and energy gains given in Fig. 5.a,b against the area overhead given in Fig. 5.c, we observe that per-CU PG provides only about 1% improvement in performance at the cost of 53% increase in the PG area overhead compared to the g16 design. On other hand, g16 provides 21% improvement in performance and 30% improvement in power efficiency at the cost of only 54% increase in PG area overhead compared to 192CU (single-cluster) granularity. Overall, we conclude that 16CU PG granularity is an optimal design choice for the studied future massively parallel GPGPU architectures. Choosing 16 CU PG granularity over per-CU granularity could reduce the die area overhead from 5-40% [4], [6] to 3-28%.

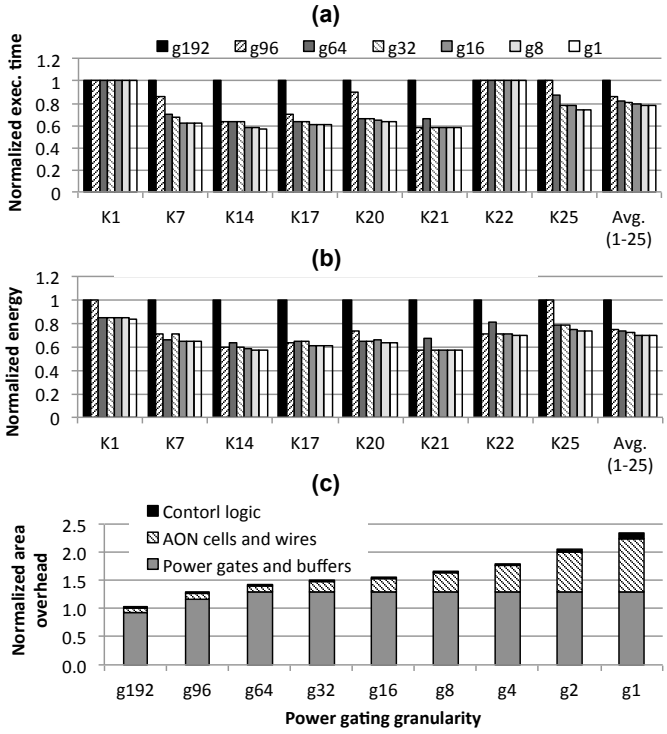

Fig. 5. a) Execution time, and b) energy of kernels at different PG granularity with TDP=150W, c) power gating area overheads at different PG granularities.

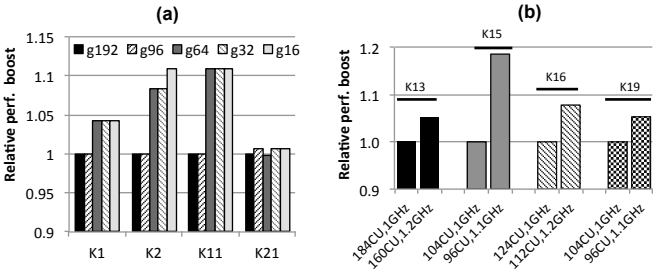

Fig. 6. Performance gain by boosting the frequency by saving idle power and using the power slack.

Observation #3: *A design-aware run-time algorithm has a better chance for frequency boosting.* This observation is especially true for less-than-TDP kernels that are frequency sensitive with relatively *flat* CU scalability around the optimal CU count. Fig. 6.a shows the performance boost of four kernels that consume less than TDP at their optimal CU count. Among them, `CoMD.EAM.advVel` (K2) and `graph500.botUpStep` (K11) have 11% performance improvement by running at higher frequency. Similarly, Fig. 6.b shows performance improvement for four kernels `kmeans.kernel_c` (K13), `GEMM.sgemmNT` (K15), `XSBench.bitSort` (K16), and `bprop.adjW` (K19) with our PG design-aware run-time algorithm in a design with 16 CU granularity. The left bars indicate performance for the kernels, where a design-decoupled runtime always enables optimal number of CUs for an application without considering any effects of PG granularity, resulting in additional CUs idle but ungated due to granularity size. These kernels do not see any frequency boosting as they reach TDP limits because of leakage power dissipated by the idle CUs. However, our design-aware run-time algorithm, as indicated by the right bars for these kernels, chooses to turn on CUs in multiples of PG granularity, and utilizes the saved idle leakage power and remaining power slack to boost frequency leading to up to 18% better performance and 5% better energy efficiency. Hence, by exposing the PG granularity as readable register or API, OS/driver/firmware can easily access such information and make runt-time decisions for additional performance gains.

Observation #4: *Design-time PG granularity is mostly independent of TDP.* So far, we have assumed a 150 W TDP in the PG analysis. Fig. 7 shows an example of such analysis for the `miniFE.matvec` kernel at 125 W, 150 W and 175 W TDP, which is a representative of a high-power kernel that can reach all TDP levels with different number of active CUs. We can see that execution time reduces as TDP increases as a result of more CUs being active. In all three TDP levels, performance starts to flatten out beyond a PG granularity of a 32 CU cluster size, and a 16 CU cluster size is good enough. Our run-time algorithm is able to adapt to the optimal CU count under different TDP constraints.

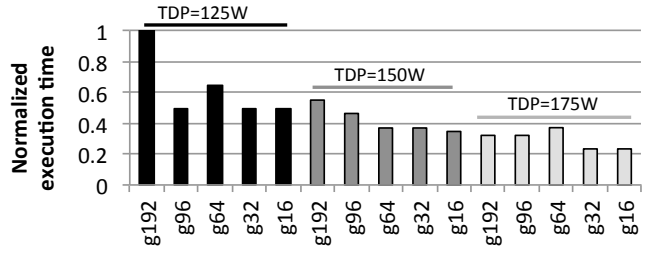

Fig. 7. Normalized execution time of *miniFE.matvec* kernel (K5) at different PG granularities and three different TDPs.

VI. Conclusions

We investigated how to leverage power gating to improve performance and power efficiency for a future massively parallel GPGPU architecture. We showed that a simplistic per-CU PG granularity only incurs significant silicon area overhead without further benefits of run-time performance. Further, to achieve better power efficiency without sacrificing performance, the design-time decision on optimal PG granularity needs to be aware of applications characteristics at run-time parallelism and vice-versa.

Acknowledgments

AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies. We wish to thank Steve Kosonocky for his advice on power gating implementation. The research work of K. Dev and S. Reda is also supported by the NSF grants 1305148 and 1438958.

References

[1] Green500-list. [Online]. Available: http://www.green500.org
[2] Exascale-project. [Online]. Available: "http://www.exascaleinitiative.org/
[3] E. Hadi *et al.*, "Dark Silicon and the End of Multicore Scaling," in *ISCA*, 2011.
[4] H. Jiang *et al.*, "Benefits and Costs of Power-gating Technique," in *ICCD*, 2005.
[5] K. Dev *et al.*, "Power Mapping and Modeling of Multi-core Processors," in *ISLPED*, 2013.
[6] S. Youngsoo *et al.*, "Power Gating: Circuits, Design Methodologies, and Best Practice for Standard-cell VLSI Designs," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 15, no. 4, pp. 28:1–28:37, Oct. 2010.
[7] B. Su *et al.*, "PPEP: Online Performance, Power, and Energy Prediction Framework and DVFS Space Exploration," in *MICRO*, 2014.
[8] X. Zhan *et al.*, "CARB: A C-State Power Management Arbiter For Latency-Critical Workloads," *IEEE Comp. Arch. Lett.*, 2016.
[9] R. Cochran *et al.*, "Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps," in *MICRO*, 2011.
[10] A. Majumdar *et al.*, "A Taxonomy of GPGPU Performance Scaling," in *IISWC*, 2015.
[11] J. Lee and N. S. Kim, "Optimizing Throughput of Power- and Thermal-constrained Multicore Processors Using DVFS and Per-core Power-gating," in *DAC*, 2009.
[12] L. Jacob *et al.*, "Power Management of Datacenter Workloads Using Per-Core Power Gating," *IEEE Comp. Arch. Lett.*, vol. 8, no. 2, pp. 48–51, Jul. 2009.
[13] I. Paul *et al.*, "Harmonia: Balancing Compute and Memory Power in High-performance GPUs," in *ISCA*, 2015.
[14] O. Kayiran *et al.*, "Neither More Nor Less: Optimizing Thread-level Parallelism for GPGPUs," in *PACT*, 2013.
[15] S. Song *et al.*, "Energy-efficient Scheduling for Memory-intensive GPGPU Workloads," in *DATE*, 2014.
[16] ITRS. [Online]. Available: http://www.itrs.net/
[17] K. Dev *et al.*, "A Framework for Evaluating Promising Power Efficiency Techniques in Future GPUs for HPC," in *HPC*, 2016.
[18] S. Kosonocky, "Practical Power Gating and Dynamic Voltage/Frequency Scaling," in *HotChips*, 2011.
[19] Mantevo-Project. [Online]. Available: http://www.mantevo.org/
[20] S. Che *et al.*, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *IISWC*, 2009.