PerDome: A Performance Model for Heterogeneous Computing Systems

Li Tang
Department of Computer Science and Engineering
University of Notre Dame
ltang@nd.edu

X. Sharon Hu
Department of Computer Science and Engineering
University of Notre Dame
shu@nd.edu

Richard F. Barrett
Center for Computing Research
Sandia National Laboratories*
rfbarre@sandia.gov

Author Keywords
Heterogeneous computing; GPU; roofline model; performance modeling; workload partitioning;

ABSTRACT

Heterogeneous systems, consisting of different types of processors, have the potential to offer higher performance at lower energy cost than homogeneous systems. However, it is rather challenging to actually achieve the high execution efficiency promised by such a system due to the larger design space and the lack of reliable performance/energy models for aiding design space exploration. This paper fills this gap by proposing a performance model for heterogeneous systems. In processor level, the roofline model [1] can produce the performance upper bound of executed code using its ratio of computation to memory traffic. Our model, referred to as PerDome, builds on the roofline model and can reliably predict the system performance for both homogeneous execution (where each processor either executes the entire application code or none) and heterogeneous execution (where each processor executes part of the application code). Two case studies are carried out to demonstrate the effectiveness of PerDome. The results show that PerDome can indeed provide a good estimate for performance comparisons which can then be used for heterogeneous system design space exploration.

INTRODUCTION

Heterogeneous systems, consisting of different types of processors, are becoming more prevalent in not only embedded computing area (e.g., Apple iPhone 6) but also high performance computing community (e.g., Oak Ridge's Titan supercomputer). Such systems have the potential to offer higher performance at lower energy cost than homogeneous systems. However, it is rather challenging for a given application to actually achieve the high execution efficiency promised by such a system. A main difficulty of efficiently using heterogeneous systems is workload partitioning, i.e., which part of an application should run on which processor. If workload partitioning can properly map different parts of an application to distinct processors of heterogeneous systems based on their characteristics, much higher performance can be achieved [2, 3]. Yet, most existing work on workload partitioning for heterogeneous systems (e.g. [2, 3, 4]) either considers partitioning in application level or assumes application partitioning is given in code level. Since for many applications it is not readily clear which partition can achieve the highest performance, design guidelines provided by a reliable performance model is indispensable.

A reliable performance model should be able to faithfully capture the relative performance levels of different workload partitioning options. We call a specific workload partitioning implementation a workload partition (WP). For homogeneous systems, the roofline model introduced in [1] is such a model and can be used to help design applications and homogeneous systems. However, straightforwardly employing the roofline model for heterogeneous systems, e.g., by simply aggregating two distinct processors' roofline plots, can only model some very specific execution scenarios and conditions.

In this paper, we propose PerDome to model both homogeneous and heterogeneous execution scenarios of heterogeneous systems. Similar to the roofline model, the PerDome model is based on the given processors' peak computation and memory performance, and operational intensity[1] values. We focus on two types of processors, CPU and GPU. The output of PerDome is performance upper bounds for different WPs on a given CPU+GPU heterogeneous system.

To validate our PerDome model, we conducted two application case studies. One is based on a synthetic application composed of a segment of compute-intensive code and a segment of memory-intensive code. The other is the data assembly stage (referred to as DA) in a representative FEM code (i.e., miniFE [5]). We implemented different WPs and measured their actual performance on different heterogeneous platforms. We then compared the actual and predicted relative performance levels of WPs for model validation.

Contributions.

The ultimate goal of this work is to help application developers partition workload on CPU+GPU heteroge-

*Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

[1]Operational intensity is defined as the ratio of arithmetic operations per byte of data moved from/to memory [1].

SpringSim-HPC, 2015, April 12 - 15, Alexandria, VA, USA

neous systems for better performance. There are two main contributions in this work.

The first contribution is the consideration of a large design space. We consider, in PerDome, two processor types and a wide variety of execution models including input date-set based partitioning, code partitioning, with and without data dependencies between the processors, etc. Particularly, we propose a novel way to represent workload partitioning options on heterogeneous systems for considering all necessary information in a simple format.

Second, we use both synthetic and representative applications to validate our model. The representative application could make our work closer to real applications. The results show that PerDome can indeed provide a good estimate for performance comparisons which can then be used to aid workload partitioning design space exploration.

BACKGROUND

In this section, we first classify different execution models of using the CPU and GPU in heterogeneous systems. We then briefly review the roofline model and discuss why simple extensions of the roofline model are insufficient for predicting the performance upper bounds of WPs for the purpose of design space exploration .

Execution Models

Given a heterogeneous system and an application kernel[2], there are different ways (i.e., execution models) to execute the kernel. In this paper, we focus on heterogeneous systems consisting of a CPU and a GPU and assume that the application kernel has multiple input data sets to operate on. A critical design consideration is how to partition the workload between the CPU and GPU. Workload here is defined as the numbers of the floating point operations to be executed (referred to as flops) and the bytes to be moved from/to main memory (referred to as bytes). Therefore, a processor's workload is dependent on both mapped kernels and input data sets. We classify the execution models of heterogeneous systems into four categories and illustrate some examples in Figure 1.

CPU-only and GPU-only are a simple way to use the processors of heterogeneous systems, where only one processor is responsible for the whole workload and the other processor is unused. These WPs can be desirable for achieving low-energy execution if one processor has much higher energy efficiency than the other [6]. In the data partitioning (DP[3]) execution model, the workload is partitioned by dividing the input data sets into two groups to be handled by the CPU and GPU separately. The granularity of the basic dividable data set unit can be

[2]Since an application may adopt different algorithms for different implementations, the application can have very different performance upper bounds for different implementations. Thus we use "kernel" to indicate that a specific algorithm choice for the application.

[3]Some papers refer to input data partitioning as workload partitioning.

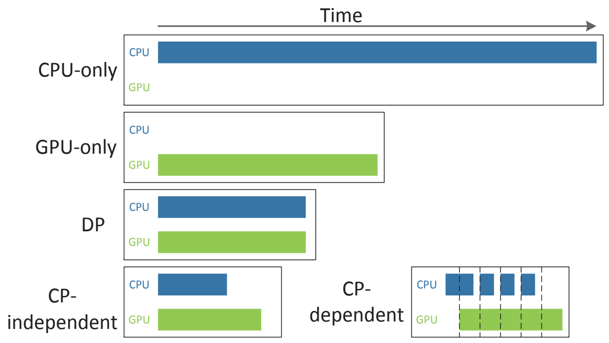


Figure 1. Four execution models of workload on heterogeneous systems. CP-independent represents the conditions that CPU and GPU execute their mapped kernel code without data communication between each other.

small and independent. Thus, perfect DP (p-DP) is assumed to be able to always finely partition the workload and distribute them evenly to CPU and GPU for perfectly overlapping the execution time of both processors. For extreme DP cases where one processor's input data sets are empty, DP is equivalent to CPU-only or GPU-only. Some recent work (e.g., [7, 8]) studied statically partitioning of input data sets or dynamically distributing tasks for balancing CPU/GPU workload.

Though DP can balance CPU/GPU workload, it cannot fully exploit the distinct capabilities of the CPU and GPU. Another execution model is code partitioning (CP) where the CPU and GPU execute different segments of the task, i.e., parts of the kernel code. Usually the more compute-intensive part of the kernel code is offloaded to the GPU for higher execution efficiency. Depending on whether CP causes CPU/GPU data dependency, we further classify CP to CP-dependent and CP-independent (see the last two execution patterns in Figure 1). The CPU and GPU of CP-independent execute their mapped kernel code concurrently without interruption for data communication. For CP-dependent, the input data sets are partitioned into small groups and the CPU and GPU communicate after processing each group of data sets instead of all the data sets. The CPU/GPU stages are scheduled by using a simple software pipelining scheme. When the number of data set groups is large enough, CP-dependent can overlap most of the execution time of CPU and GPU. However, CP-dependent becomes much more complicated and incurs high synchronization overhead when there are more than two CPU/GPU stages. Thus we only focus on two CPU/GPU stages in our performance model.

Roofline Model

Williams et al. introduced the roofline model in [1] to derive the performance upper bound for a given processor based on the processor's peak computation performance, memory bandwidth, and the operational intensity of any kernel code. The performance bottleneck of a kernel code can be easily obtained by using the kernel's operational intensity to find the corresponding performance upper bound in the roofline plot.

Let I represent the operational intensity, and is defined by

$$I = \frac{N_f}{N_b}, \tag{1}$$

where N_f and N_b are the number of floating point operations (flops) and the number of bytes moving to/from the main memory (bytes), respectively. The assumption behind the roofline model is that processors can perfectly overlap the computation and memory traffic. Hence, the processor execution time is modeled as

$$\begin{aligned} T &= \max\{N_f t_f, N_b t_b\} \\ &= N_b t_b \max\{I \frac{t_f}{t_b}, 1\}, \end{aligned} \tag{2}$$

where t_f and t_b are the average time per flop and byte movement. The performance upper bound, P, using flops/sec as the unit is calculated by

$$\begin{aligned} P &= \frac{N_f}{N_b t_b \max\{I \frac{t_f}{t_b}, 1\}} \\ &= \frac{1}{t_b \max\{\frac{t_f}{t_b}, I^{-1}\}}. \end{aligned} \tag{3}$$

Since t_f and t_b can be treated as constants for a particular processor, Equation (3) indicates that the processor is memory bounded if $I\frac{t_b}{t_f}$ is smaller than I. The roofline plot depicts this performance trend in a 2D space and uses x- and y-axes for kernel's operation intensity and performance upper bound, respectively.

The roofline model can be readily extended to predict performance upper bounds for homogeneous systems. That is, the system performance upper bound is simply the aggregation of all processors' performance upper bounds since each processor is identical and the best WP is evenly distributing data sets to all processors. The roofline model can also be easily applied to *some* execution models in heterogeneous systems. For CPU-only and GPU-only, the roofline model can be straightforwardly used to predict the system performance upper bound. In p-DP, one can add the performance upper bounds of CPU and GPU to get the system performance upper bound since CPU and GPU concurrently run the whole kernel from beginning to end.

However, for CP, it is not immediately clear how to apply the roofline model since CP can have many different WPs and their CPU and GPU might have different intensities and execution times. One possibility is to aggregate the performance upper bounds of CPU and GPU with their corresponding intensities to get the system performance upper bound. But this method can lead to upper bounds that are too loose since it does not take into consideration that CPU and GPU may finish at different times. For example, consider a CP based WP having its CPU's execution time much longer than its GPU's. By using separate CPU and GPU intensities and the roofline model, the performance upper bounds of CPU and GPU, P_C and P_G, can be obtained. Assuming that $P_C << P_G$, simply aggregating P_C and P_G would result in a system upper bound much higher than the actual system performance. A tighter bound for this WP should be close to P_C since the GPU is not used for most of the system execution time. Having upper bounds that are too loose often causes misleading performance comparison results.

Overall, a good performance model for heterogeneous systems should be able to handle all the execution models and result in performance upper bounds that are tight and have high fidelity. By high fidelity, we mean that the relative magnitude of the performance upper bounds should mostly reflect the relative magnitude of the corresponding actual performance values. To our best knowledge, no such models exist for heterogeneous systems.

PERDOME MODEL

In this section, we first define three essential WP parameters, which can uniquely represent a WP on heterogeneous systems for the purpose of performance prediction. We then present the details of the PerDome model based on these parameters.

Essential Workload Partition Parameters

The essential WP parameters are the input to the PerDome model for generating performance upper bounds. There are two key considerations for selecting appropriate parameters as the essential WP parameters. First, the parameters should cover the whole workload partitioning design space and each WP should have a unique representation by using the parameters. Second, the number of such parameters should be small so that the resulting model is easy to use.

For a given heterogeneous system, an application kernel, and any given WP, according to the workload definition, $\{N_f,\ N_b\}$, $\{N_{f-C},\ N_{b-C}\}$ and $\{N_{f-G},\ N_{b-G}\}$ uniquely represent the workload of the entire application, and the workloads (numbers of flops and bytes) of the CPU and GPU, respectively. The actual values of these workload parameters are compiler and architecture dependent, but can be estimated at the algorithm level as discussed in [1]. Timing parameters (time per flop and byte), $\{t_{f-C},\ t_{b-C}\}$ and $\{t_{f-G},\ t_{b-G}\}$, reflect the peak computation and memory performance of the CPU and GPU, respectively, and can be obtained by using processors' specification data and running memory benchmarks.

For heterogeneous systems, the execution time is dominated by the slower processor. Based on the parameters introduced above and the concept used in Equations (2-3), the upper bound on the system performance can be calculated as

$$P = \frac{N_f}{T} \tag{4}$$

where

$$T = \max\{\max\{N_{f-C}t_{f-C}, N_{b-C}t_{b-C}\}, \max\{N_{f-G}t_{f-G}, N_{b-G}t_{b-G}\}\}. \tag{5}$$

Since the timing parameters are constant for a given hardware platform, to estimate performance upper bound, a straightforward way is to use all 6 WP parameters, i.e., N_f, N_b, N_{f-C}, N_{b-C}, N_{f-G} and N_{b-G}. However, as discussed in [1], it is more general to use operational intensities instead of raw flops and bytes.

The roofline model [1] uses a single kernel's operational intensity as its input to model performance of workload on one processor. Thus, a straightforward extension for the inputs of the PerDome model is using both the operational intensity for the CPU and that of the GPU. This choice does allow the PerDome model to cover the whole workload partitioning design space. However, it does not take into consideration the operational intensity of the entire application kernel. Given two different application kernels that have different operational intensities, it is possible for them to have WPs with exactly the same CPU and GPU intensities. If only CPU and GPU intensities are used to predict performance upper bound, the two application kernels would have the same upper bound. Now suppose one application kernel executes most of its code on the CPU while the other application kernel executes most of its code on the GPU. It is highly likely that the two application kernels have significantly different performance upper bounds. Thus, simply using the operational intensities of the CPU and GPU is not sufficient.

Let $\{I_C, I_G\}$, defined in Equations (6-7) below, be the operational intensities of the kernels assigned to CPU and GPU, respectively.

$$I_C = \frac{N_{f-C}}{N_{b-C}} \tag{6}$$

$$I_G = \frac{N_{f-G}}{N_{b-G}} \tag{7}$$

We choose to use I, I_C and I_G as the essential WP parameters for the PerDome model, where I is the operational intensity of the whole application kernel. Note, the whole application kernel is composed of the kernels assigned to CPU and GPU, and the operational intensity is zero when the corresponding processor is unused. Clearly, each WP can be represented by a particular set of the essential WP parameters.

Below, we present two observations which are useful for comparing performance upperbounds of different WPs.

LEMMA 1. *For an application kernel having intensity I, if a WP satisfies $I_C = I$ (resp., $I_G = I$), then either $I_G = I$ (resp., $I_C = I$) or $I_G = 0$ (resp., $I_C = I$).*

Proof: Note that the $I_G = 0$ and $I_C = 0$ cases correspond to the CPU-only and CPU-only WP, respectively. Hence the lemma holds. For the case of $I = I_G = I_C$, by definition, we have

$$N_f = N_{f-C} + N_{f-G}, \tag{8}$$

and

$$N_b = N_{b-C} + N_{b-G} \tag{9}$$

By combining Equations (6-7) and using the condition that $N_{f-C}/N_{b-C} = N_f/N_b$ we get

$$N_{f-C} \times N_b = N_{f-C} \times N_{b-C} + N_{f-G} \times N_{b-C} \tag{10}$$

With some manipulations, we get $N_{f-G}/N_{b-G} = N_f/N_b$. ■

LEMMA 2. *For an application kernel having intensity I, if a WP satisfies $I_C > I$ (resp., $I_G > I$), then $I_G < I$ (resp., $I_G > I$).*

This lemma can be proved in a similar fashion but with more mathematical manipulation. We omit the details due to space limit. Lemma 2 indicates that a kernel cannot be partitioned into two smaller kernels with their operational intensities both higher or lower than the original kernel's operational intensity.

Performance Upper Bound

We now derive the performance upper bound expression using the essential WP parameters. Note the performance upper bound defined in Equation (4) can be rewritten as

$$P = \max\{\frac{N_{f-C}}{N_f}t_{f-C}, \frac{N_{b-C}}{N_f}t_{b-C}, \frac{N_{f-G}}{N_f}t_{f-G}, \frac{N_{b-G}}{N_f}t_{b-G}\}^{-1} \tag{11}$$

Our goal is to express N_{f-C}, N_{b-C}, N_{f-G} and N_{b-G} as functions of the essential WP parameters, $\{I, I_C, I_G\}$. We consider two cases: (i) $I \neq I_C \neq I_G$ and (ii) $I = I_C = I_G$. According to Lemma 1 and 2, these two cases cover all the possible WPs for a given application kernel.

Consider first $I \neq I_C \neq I_G$. According to the definition of operational intensity in Equation (1), we have

$$I * N_b = I_C * N_{b-C} + I_G * N_{b-G}. \tag{12}$$

The total number of moved bytes, N_b, is equal to the sum of N_{b-C} and N_{b-G}. Thus, we can extend Equation (12) to

$$I * (N_{b-C} + N_{b-G}) = I_C * N_{b-C} + I_G * N_{b-G}. \tag{13}$$

By incorporating the essential WP parameters, $\{I, I_C, I_G\}$, we can solve Equation (13) to get the ratio of N_{b-C} to N_{b-G}

$$\frac{N_{b-C}}{N_{b-G}} = \frac{I_G - I}{I - I_C}. \tag{14}$$

By replacing N_{b-C} by $N_b - N_{b-G}$ in Equation (14), we have

$$N_{b-G} = N_b\frac{I - I_C}{I_G - I_C}. \tag{15}$$

Then N_{b-C} can be obtained by solving $N_b - N_{b-C}$

$$N_{b-C} = N_b\frac{I - I_G}{I_C - I_G}. \tag{16}$$

Similarly, we can get the numbers of flops for CPU and GPU

$$N_{f-G} = I_G N_b \frac{I - I_C}{I_G - I_C}. \tag{17}$$

$$N_{f-C} = I_C N_b \frac{I - I_G}{I_C - I_G}. \tag{18}$$

By using Equations (15-18) and the performance upper bound expression in Equation (11), we have

$$\begin{aligned} P = \max\{ & t_{f-C}\frac{I_C(I - I_G)}{I(I_C - I_G)}, t_{b-C}\frac{I - I_G}{I(I_C - I_G)}, \\ & t_{f-G}\frac{I_G(I - I_C)}{I(I_G - I_C)}, t_{b-G}\frac{I - I_C}{I(I_G - I_C)}\}^{-1}. \end{aligned} \tag{19}$$

Based on the performance upper bound expression in Equation (19), we construct our PerDome model. For ease of visualization, each PerDome plot corresponds to a specific application kernel intensity I value. This choice allows the user of the model to easily compare how different WPs (I_C and I_G) impact the system performance. A PerDome plot thus forms a surface in the 3-D space where the x- and y-axes depict the operational intensities of the kernels assigned to CPU and GPU, and the z-axes depicts the corresponding performance upper bound. Consequently, for a given application kernel's operational intensity, the PerDome model readily reveals which combination of the CPU and GPU intensities leads to the highest performance upper bound.

Now consider $I = I_C = I_G$. In this case, we have $N_{f-C} = I \times N_{b-C}$ and $N_{f-G} = I \times N_{b-G}$. Using Equation (9), we can re-write the performance upper bound as

$$\begin{aligned} P = \max\{ & (\frac{N_{f-C}}{N_f}t_{f-C}, \frac{N_{f-C}}{N_f \times I}t_{b-C}, \\ & (1 - \frac{N_{f-C}}{N_f})t_{f-G}, (1 - \frac{N_{f-C}}{N_f \times I})t_{b-G}\}^{-1} \end{aligned} \tag{20}$$

For a given application kernel intensity I value, the performance upper bound in this case depends on the ratio of N_{f-C} to N_f. In general, the ratio is chosen such that the highest performance upper bound is achieved. Therefore, in this case there is only a single point for the PerDome plot with $I_C = I_G$.

Note that Equation (20) can also be used to obtain the performance upper bound for the cases of $I_G = 0$ and $I_C = 0$ (i.e., the CPU-only and GPU-only WPs) by simply setting N_{f-C} to N_f or 0, respectively. Thus, the performance upper bounds for these two cases correspond to two points in the PerDome plot with $I_G = 0$ and $I_C = 0$, respectively.

EXPERIMENTAL VALIDATION

In this section, we present our effort in validating the PerDome model. We first briefly discuss the case studies. We then give details on the hardware platforms used and their associated timing parameters. Lastly, we apply the PerDome model to the two case studies and compare the trends of different WPs' actual performance and their performance upper bounds for model validation.

Case Studies

We used two case studies in our validation effort: one is a synthetic application kernel and the other is the DA kernel in miniFE [5]. A number of WPs based on different execution models are considered for each application kernel.

Synthetic Application Kernel

Since a synthetic application (SA) kernel allows one to easily control the numbers of flops and bytes in the whole application kernel as well as in the kernels assigned to CPU and GPU, we use it to help explore different WP options. To show the benefits of different WPs, we construct the SA kernel with a piece of compute-intensive code and a piece of memory-intensive code. The main component of the SA kernel is shown below.

```
#define SIZE 2560000

float a[8][SIZE], b[SIZE], c[SIZE], d[SIZE], e[SIZE];

for (int i=0;i<SIZE;i++) {
   for (int j=0;j<8;j++) b[i] += pow(a[j][i],16); \\
       PowAdd: 128 flops & 8 bytes
   e[i] = c[i] + d[i]; \\ VecAdd: 1 flop & 12 bytes
}
```

The input to the SA kernel are vectors `a`, `c` and `d`. The output of the SA kernel are vectors `b` and `e`. To stress the CPU and GPU usage, we set `SIZE` to be 2560000. The main loop iteratively runs two functions: PowAdd and VecAdd. PowAdd is to aggregate the sixteenth power of data items in `a` vector and has the operational intensity of 2. VecAdd is a regular vector addition of two vectors and has the operational intensity of 0.083.

We implemented four WPs based on the classified execution models. For the CPU-only WP, the implementation employs the OpenMP static scheduling to evenly distribute the main loop iterations to available threads. For the GPU-only WP, each iteration is assigned to one GPU thread. The WP of p-DP divides the main loop iterations into two parts. One runs the CPU-only executable while the other runs the GPU-only executable. The partitioning is done such that the CPU and GPU complete the execution at the the same time. Since the PowAdd and VecAdd functions in the SA kernel are data independent, we implemented the CP-independent WP, where all main loop iterations of PowAdd are run on GPU and VecAdd are on CPU.

DA in miniFE

To demonstrate how the PerDome model can help in application development, we selected the DA stage in miniFE as our second case study. As discussed in [5], miniFE is sufficiently complex and is a proxy FEM application that could be used to predict the performance trends of real-world FEM applications. The main role of DA is to generate an equation system including a large sparse global matrix stored in the compressed sparse row (CSR) format. The original implementation of DA in miniFE has two separate functions: (i) stiffness matrix computation (STC) and (ii) stiffness matrix assembly (SMA). More details of DA in miniFE can be found in [8].

To implement the CPU-only and GPU-only WPs of DA, we follow the ideas in [8] and select `Omp` and `SmemDA` as the implementations of DA on CPU and GPU, respectively. `Omp` uses the OpenMP static scheduling to distribute the workload onto each thread evenly at the element level. In `SmemDA`, the work of one element is assigned to eight GPU threads. For the p-DP WP, we partition the workload at the element level following the performance ratio of `Omp` to `SmemDA` and concurrently run `Omp` and `SmemDA` with their corresponding elements on CPU and GPU, respectively.

Given the more complex structure of DA, different CP based WPs can be derived but it is not always clear which of these WPs would have higher performance. For our case study, we implemented two CP-dependent WPs. The first one, referred to as CP1, only offloads STC onto the GPU. The rationale behind this WP is that SMA in DA is mainly composed of irregular memory accesses and STC is mainly responsible for matrix computation. Another CP-dependent WP, referred to as CP2, tries to further move some GPU unfriendly code out of STC and offload it onto the GPU. STC has some operations, e.g., division and branching operations, that are not efficient for GPU execution. In CP2, we only offload the key computation part of STC onto the GPU. Since both CP1 and CP2 are CP-dependent, we employ the simple software pipelining scheme to schedule the CPU and GPU stages in CP1 and CP2.

Model Parameters

We use two CPUs and two GPUs to form four heterogeneous systems.. The hardware details are summarized in Table 1. The problem sizes of the SA and DA kernels are 2560000 and 100^3, respectively. The data type is single precision floating point.

We use the processors' architecture information, such as the number of cores and core base frequencies (the feature of multiple issue is not used in real execution), to compute peak computation performance. The values of t_{f-C} or t_{f-G} are set to the inverse of the corresponding processors' peak computation performance and are shown in Table 1. For the memory timing parameters, the STREAM [9] and SHOC [10] benchmarks are used to measure the peak memory bandwidth of our CPUs and GPUs. The peak memory bandwidth is used to get t_{b-C} and t_{b-G} which are summarized in Table 1.

We compute the essential WP parameters of the WPs for the SA and DA kernels and summarize them in Table 2. Here, columns 2, 3 and 4 show the operational intensities of the application kernel, kernels assigned to CPU and GPU, respectively.

Model Validation

Table 1. Processor data

Processor	# of cores	Base core clock (GHz)	Time per flop (ps)	Time per byte (ps)
i7 2600K	4	3.4	73.5	65.9
i3 2100T	2	2.5	200	73
GTX Titan	2688	837	0.4	4.1
GTX 750	512	1020	1.9	14.8

Table 2. Essential WP parameters

Workload partition	I	I_C	I_G
CPU-only (DA)	4.4	4.4	0
GPU-only (DA)	4.4	0	4.4
p-DP (DA)	4.4	4.4	4.4
CP1 (DA)	4.4	0.4	5.4
CP2 (DA)	4.4	1.5	9.2
CPU-only (SA)	1.7	1.7	0
GPU-only (SA)	1.7	0	1.7
p-DP (SA)	1.7	1.7	1.7
CP (SA)	1.7	0.1	2.0

We first use Equation (19) to calculate the performance upper bounds for all combinations of I_C and I_G values for SA and DA on our selected heterogeneous systems, i.e., i7+GTX750, i7+Titan, i3+GTX750 and i3+Titan. The timing and essential WP parameters needed by Equation (19) are given in Tables 1 and 2. Figures 2(a)-(d) and Figure 3(a)-(d) depict the performance upper bounds for SA and DA running on the four heterogeneous platforms, respectively. In all the figures, the x-, y- and z-axes represent I_C and I_G, and performance upper bounds, respectively. Both the x- and y-axes use logarithmic scale with base of 2. The graphs in these figures share some common features. Specifically, there are two separate surfaces representing performance upper bounds of CP based WPs. These WPs are in two distinct regions of combinations of I_C and I_G, which are shaded in grey in the x-y plane. The other two regions are blank since their I_C and I_G combinations have the I_C and I_G values both larger or smaller than the original application's operational intensity (which would never exist according to Lemma 2). The two surfaces have high performance upper bounds when I_G approaches 1.7 and 4.4 for SA and DA, respectively. Each figure also includes a point corresponding to p-DP running on the corresponding hardware platform. The performance upper bounds of p-DP is the aggregation of the ones of CPU-only and GPU-only. The points of CPU-only and GPU-only are omitted since p-DP has higher performance upper bounds than CPU-only and GPU-only.

We next study the tightness of the generated performance upper bounds. Based on the PerDome plots, performance upper bounds can be obtained by checking the corresponding combination of I_C and $I_G$values. The measured actual performance (as bars) with the corresponding predicted performance upper bounds (as markers) of SA and DA are shown in Figures 2(e) and 3(e), respectively. It can be seen that, for SA, the actual performance values of p-DP and CP are close to the predicted performance upper bounds. For example, the actual performance of CP on i3+Titan achieves 97% of the predicted performance upper bound. On the other hand, for DA, the actual performance values of WPs are significantly lower than the corresponding predicted performance upper bounds due to DA's complicated computation and memory behaviors. CP1 on i3+GTX750 has the tightest predicted performance upper bound among the others and its actual performance only achieves 43% of the predicted performance upper bound. Thus, our PerDome can generate tight performance upper bounds for applications with simple computation and memory behaviors that can fully exploit the processor power, but only provides relatively loose bounds for other cases.

To examine the fidelity of the predicted performance upper bounds, we compare the actual relative performance levels with the predicted ones for different WPs. As Figure 2(e) shows, for SA, the actual performance trends completely agree with the predicted performance upper bound trends. For example, CP has 14% and 17% higher predicted performance upper bounds and 3% and 9% higher actual performance than p-DP on i7+GTX750 and i3+GTX750, respectively. Also, the performance upper bounds of CP are 61% and 65% lower than p-DP, and the actual performance values of CP are 60% and 62% lower than p-DP on i7+Titan and i3+Titan, respectively.

Figure 3(e) also confirms the high fidelity of our PerDome model for DA. Specially, on i7+GTX750, CP1 has 7% higher predicted performance upper bound and 6% higher actual performance than p-DP. On all the other systems, the actual and predicted relative performance levels agree with that p-DP is faster than CP1 and CP1 is faster than CP2. Thus, the predicted performance upper bounds can reliably reflect the relative magnitude of the corresponding actual performance.

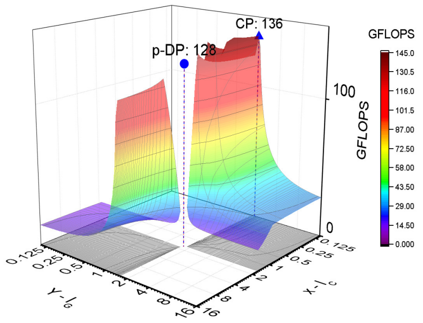
Using the PerDome plots, one can get some useful design guidelines. Take Figure 2(a) as an example. This figure is for executing SA on i7+GTX750. One can observe that the point corresponding to the p-DP WP has a performance upper bound (128 GFLOPS) that is lower than the top part of the right surface, which indicates that the CP based WPs in this region may have higher actual performance than p-DP. Furthermore, we can see that CP has higher actual performance upper bound (136 GFLOPS) than p-DP.

SUMMARY

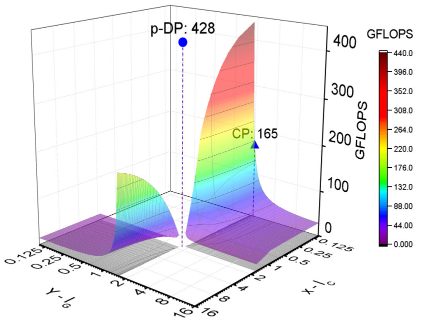
We have proposed a performance model, PerDome, for heterogeneous systems by extending the roofline model introduced in [1]. PerDome is applicable to a variety of execution models and uses a minimal set of WP parameters. To validate the effectiveness of our proposed model, we carried out two detailed case studies. Specially, we implemented a number of different WPs for the SA and DA kernels on four heterogeneous systems, and measured the actual performance of these implementations. Comparisons between the actual performance and the predicted performance upper bounds show that the PerDome model can generate tight performance upper bounds and provide reliable relative performance levels for heterogeneous systems.

REFERENCES

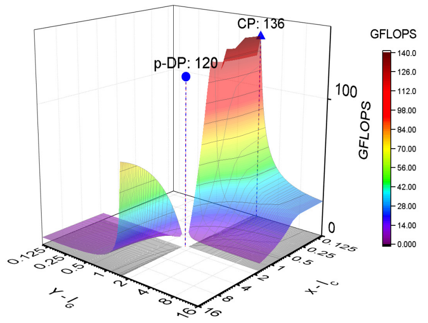
1. S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.
2. W. Sodsong, J. Hong, S. Chung, Y. Lim, S.-D. Kim, and B. Burgstaller, "Dynamic partitioning-based jpeg decompression on heterogeneous multicore architectures," in *PMAM*. ACM, 2014, p. 80.
3. M. Daga and M. Nutter, "Exploiting coarse-grained parallelism in b+ tree searches on an apu," in *SCC*. IEEE, 2012, pp. 240–247.
4. K. Ma, X. Li, W. Chen, C. Zhang, and X. Wang, "Greengpu: A holistic approach to energy efficiency in gpu-cpu heterogeneous architectures," in *ICPP-41*. IEEE, 2012, pp. 48–57.
5. R. Barrett, P. Crozier, D. Doerfler, M. Heroux, P. Lin, H. Thornquist, T. Trucano, and C. Vaughan, "Assessing the Validity of the Role of Mini-Applications in Predicting Key Performance Characteristics of Scientific and Engineering Applications," *Journal of Parallel and Distributed Computing*, vol. 75, 2014.
6. L. Tang, X. S. Hu, and R. F. Barrett, "Performance and energy implications for heterogeneous computing systems: A minife case study," *Sandia National Laboratories, Tech. Rep. SAND2014-20215*, 2014.
7. C.-K. Luk, S. Hong, and H. Kim, "Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," in *MICRO-42*. IEEE, 2009, pp. 45–55.
8. L. Tang, X. S. Hu, D. Z. Chen, M. Niemier, R. F. Barrett, S. D. Hammond, and G. Hsieh, "Gpu acceleration of data assembly in finite element methods and its energy implications," in *ASAP-24*. IEEE, 2013, pp. 321–328.
9. J. D. McCalpin, "Stream: Sustainable memory bandwidth in high performance computers," Tech. Rep., 1991-2007.
10. A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The scalable heterogeneous computing (shoc) benchmark suite," in *GPGPU-3*. ACM, 2010, pp. 63–74.



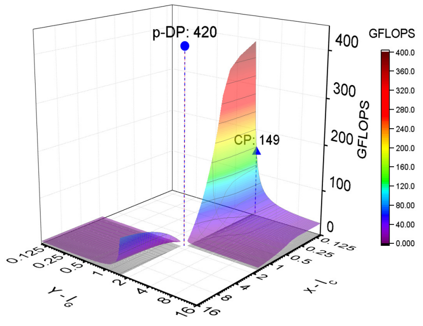
(a) i7+GTX750



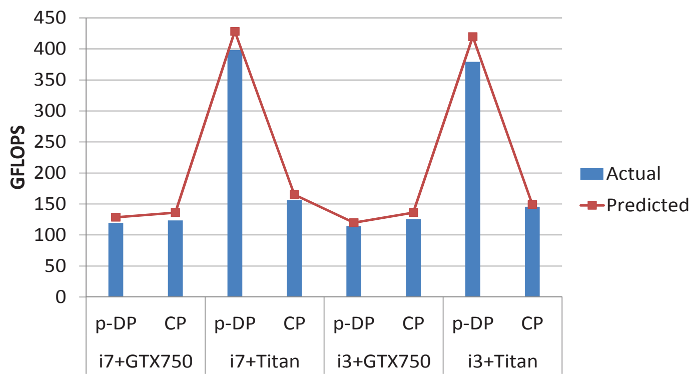
(b) i7+Titan



(c) i3+GTX750

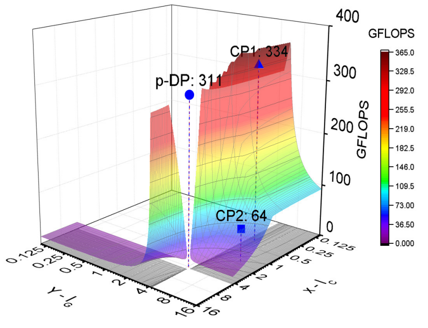


(d) i3+Titan

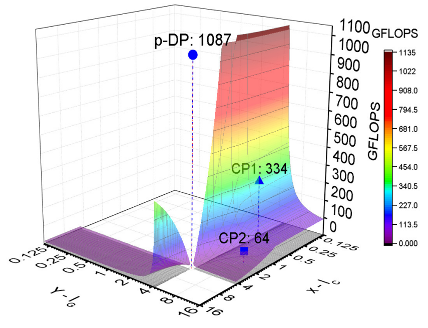


(e) Actual time

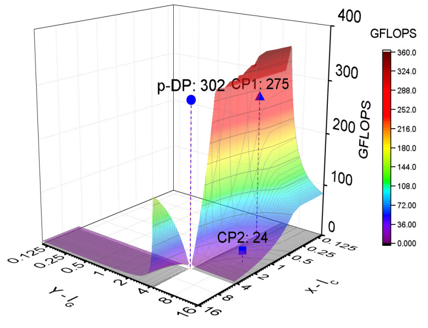
Figure 2. Performance upper bounds and actual execution time of executing SA on four different platforms.



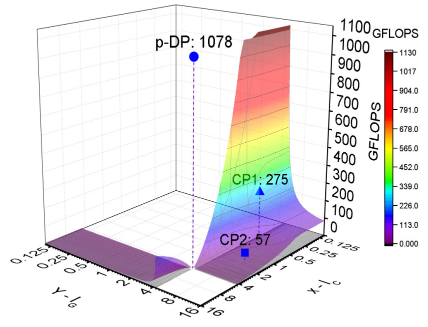
(a) i7+GTX750



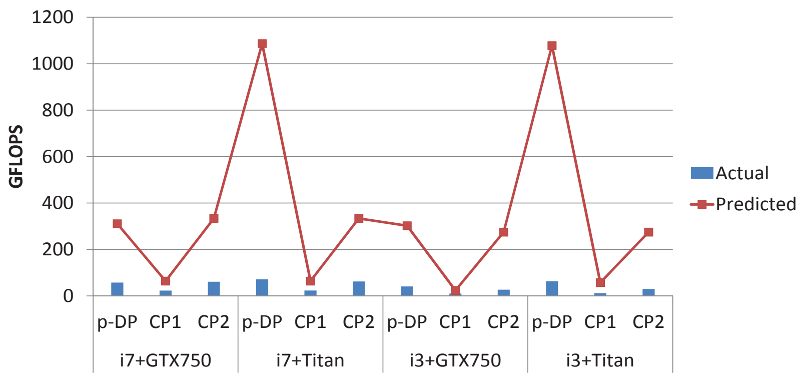
(b) i7+Titan



(c) i3+GTX750



(d) i3+Titan



(e) Actual time

Figure 3. Performance upper bounds and actual execution time of executing DA on four different platforms.