Queueing Theory Modeling of a CPU-GPU System

Lindsay B.H. May, Robert J. Voigt
Northrop Grumman Corporation, Electronic Systems Sector
Lindsay.May@ngc.com, Robert.Voigt@ngc.com
May 11, 2010

There are many tools for modeling compute system performance where there is either a single processor or a homogeneous group of parallel processors [1]. There have been few attempts to model the multi-core heterogeneous systems that are becoming more prevalent today. Some models, such as the Kuck diagram, provide a graphical and conceptual model of the processor/memory architecture but do not attempt to model the performance of such systems. For many of these systems, there is a common theme of a central processing unit (CPU) with some memory that queues jobs and/or data to be served. For multi-core systems, there are multiple servers, or processing elements, along with a supporting memory architecture. Queuing theory appears to map well onto these architectures providing for the heterogeneity that is needed and a hierarchy to support multi-tiered heterogeneous processing. We apply the graphical depiction of the Kuck diagram along with a queuing model of the processor/memory architecture to quantitatively predict the performance of a variety of architectures using specifications of memory sizes and bandwidth. The focus of this initial study is on the relationship between a CPU and a Graphics Processing Unit (GPU). Our goal, in addition to a providing a predictive model for these types of architectures, is to validate the model against actual CPU-GPU based systems.

The Kuck Model

The Kuck model [4], depicted in Figure 1, is a general parallel machine model that utilizes processing clusters organized in a hierarchical fashion. Each independent memory unit is attached to a processor containing registers and caches; a network connects the components to provide interprocessor communication and indirect memory access but does not allow for shared memory addressing. The shared-memory network attaches the processors directly to the shared-memory address space. Thus, there are two types of memory access possible in this model: fast direct memory access to the shared memory and slower indirect memory access. Further, clusters of memory units, processors, and a network are combined via a hierarchy of networks, shared-memory, and shared-memory networks. Kuck explains that parallel machines must be exploited both horizontally, via parallel processing, and vertically, via memory-hierarchy management.

In addition to the conceptual, easily visualized model, Kuck develops an abstract performance model for the hierarchical system model which can be used to qualitatively compare various architectures. He acknowledges that practical use would be difficult due to the problem of separating and measuring the factors that determine system performance. Motivated by the desire to achieve a quantitative comparison, we consider the parallel machine model from the perspective of queueing theory.

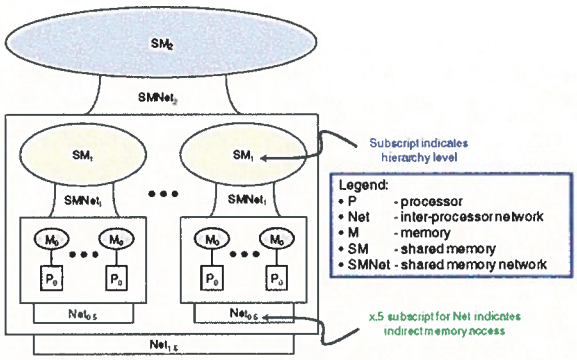

Figure 1: Kuck provides a method for describing a hardware architecture and a memory and communication hierarchy

Queueing Theory and the Queueing Model of the CPU-GPU System

Queueing theory provides a mathematical description of customers arriving for service, waiting in lines, and subsequently receiving service. Further, using queueing theory, one can calculate performance measures for a system in steady state such as the expected number of customers in a queue and in a system, the expected waiting time in a queue, the expected time spent in a system, and the probability that a system is in a specified state. There are two important factors that influence the formation of a queue: average rate at which demands are made and the statistical fluctuation about the mean arrival rate. Queues form even when the average rate is less than the system capacity as a result of intermittent arrivals [3]. While there are many queueing models, we will discuss only those relevant to our CPU-GPU queueing system model.

In terms of a CPU or GPU, we equate memory with queues and the CPU/GPU cores with servers; see Figure 2. Note that we consider three subsystems: the CPU, the GPU global memory, and the Compute Units (each compute unit is composed of eight GPU cores). In order to develop the equations for performance measures [2] of a CPU-GPU system, we consider the equilibrium solution to a continuous-time Markov chain model. The variables of interest in this system include λ, the arrival rate to the system; μ, the service rate at the server; c, the number of servers; and the ratio $\rho = \lambda/(c\mu)$, which is also the server utilization. The M/M/1 system, which we use to model the CPU core, exhibits Markovian (exponential) interarrival and service time distributions, has one server, and allows for an unlimited queue length. Then the probability, p_k, of finding k customers in the system is given by $p_k=(1-\rho)\rho^k$. The expected number of customers in the system (in steady state) is given by $L=\rho/(1-\rho)$ and the number of customers in the queue is calculated using $L_q=\rho^2/(1-\rho)$. W denotes the expected amount of time a customer spends in the system, including service, and is given by $W=1/(\lambda-\mu)$. The expected

waiting time (time in queue only) is calculated using the formula $W_q=\rho W$.

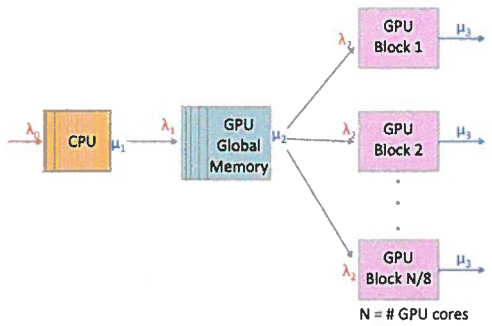

Figure 2: Open system of queueing models

Also of interest are queueing models in which the queue capacities are limited and the total number of queueing slots available is given by K-c, where K is the total system capacity. In this case, we would study an M/M/c/K queueing model. In our work, we are interested in two sub cases of the M/M/c/K model, namely the M/M/1/K model, with which we describe the single-server GPU global memory, and the M/M/c/c model, in which the system capacity equals the number of servers and there is therefore no queueing. The latter model describes the Compute Units. The job size for the system equals the size of the shared memory at a compute unit. In the case of the M/M/1/K model for $0\le n\le K$ and the M/M/c/c model for $0\le n\le c$, respectively, we have

$$p_n=\begin{cases}\frac{(1-\rho)\rho^n}{1-\rho^{K+1}}, & (\rho\neq 1)\\ \frac{\rho^n}{K+1}, & (\rho=1)\end{cases} \text{ and } p_n=\frac{\rho^n/n!}{\sum_{i=0}^{c}\rho^i/i!}.$$

For both of these special cases, we calculate the other four performance measures using the following equations:

$$L_q=\frac{p_0\rho^{c+1}/c}{c!(1-(\rho/c))^2}\left[1-(\rho/c)^{K-c+1}-(1-(\rho/c))(K-c+1)(\rho/c)^{K-c}\right],$$

$$L=L_q+c-p_0\sum_{n=0}^{c-1}\frac{(c-n)\rho^n}{n!},$$

$$W=\frac{L}{\lambda(1-p_k)}, \text{ and } W=\frac{L_q}{\lambda(1-p_k)}.$$

In general, when there are a finite number of queue spaces and they are all filled and an additional customer arrives, that customer is either turned away and lost forever from the system or returned to the calling population.

Inputs to the Queueing Model

It would be convenient if we could simply combine these three subsystems and determine the behavior of the aggregate system by adding the performance measures of the subsystems. However, before we can determine the aggregate behavior, we must describe the connections between the subsystems. The arrival and service rates depend directly on CPU and GPU specifications of the components of the architecture. The arrival rate at the CPU (λ_0) is determined by typical problem parameters we would encounter in a real application. The service rate (μ_1) depends on the bandwidth between the CPU and GPU global memory. The arrival rate at the GPU global memory (λ_1) involves the service rate at the CPU and a factor describing parallelizability of the algorithm, since GPUs are appropriate for highly parallelizable algorithms. We describe the service rate at the GPU global memory (μ_2) as a function of bandwidth between GPU global memory and the GPU cores, a value determined by the specific choice of GPU in the architecture. Finally, the arrival rate at a CU (λ_2) depends on an appropriate fraction of the service rate at the GPU and a factor describing data reusability. The service rate at the CU (μ_3) depends on the GPU shader clock speed.

We also need to account for arrivals at the CU when there are no queueing spaces available and jobs cannot be lost or sent back to the calling population. Namely, if no CUs are open to accept jobs, the GPU global memory subsystem must idle (stall) until there is an available space for a job. Similarly, if the GPU global memory queue becomes full, the CPU would have to idle and refrain from processing any jobs until a space opens in the GPU global memory queue. To address this concern, we require that the service rates at the CPU and GPU global memory depend on the number of jobs in the CU and GPU global memory subsystems, respectively. Therefore, if the CUs are all occupied servicing a job, the GPU will have a service rate of zero until a server is available at the CUs. This feedback allows us to integrate the three subsystems and subsequently calculate performance measures for the aggregate system. Specifically, we combine the input job size with the output expected time in system (W) to determine the throughput in bytes per second, which we can equate to FLOPS. Finally, we compare the FLOPS performance prediction to measurements taken on real GPU systems.

Summary

This study is a work in progress. Thus far, we have conducted initial trials on several specific GPU systems and believe that we have a solid foundation to be able to quantitatively predict their performance. We have implemented the model in MATLAB and run multiple data sets to build a comparison between the model results and several hardware platforms in the laboratory. The model will give us the expected delays of the system and the average queue lengths. Once we have obtained a statistically significant data set, we can determine the mapping between throughput and operations per second. The preliminary numbers indicate a good correlation between the model and the measured hardware data.

References

[2] D.E. Culler et al., "LogP: a practical model of parallel computation," *Communications of the ACM*, Vol. 39, No. 11, Nov. 1996, 78-85.

[2] D. Gross, C.M. Harris, *Fundamentals of Queueing Theory,* Second Edition, John Wiley & Sons, 1985.

[3] L. Kleinrock, *Queueing Systems, Volume I: Theory,* John Wiley & Sons, 1975.

[4] D.J. Kuck, *High Performance Computing: Challenges for Future Systems,* Oxford University Press, 1996.