American Journal of Networks and Communications
2015; 4(3): 67-74
Published online June 8, 2015 (http://www.sciencepublishinggroup.com/j/ajnc)
doi: 10.11648/j.ajnc.20150403.18
ISSN: 2326-893X (Print); ISSN: 2326-8964 (Online)

Performance Analysis of CPU-GPU Cluster Architectures

Ho Khanh Lam

Faculty of Information Technology, Hung Yen University of Technology and Education, Hung Yen, Vietnam

Email address:
lamhokhanh@gmail.com

To cite this article:
Ho Khanh Lam. Performance Analysis of CPU-GPU Cluster Architectures. *American Journal of Networks and Communications*. Vol. 4, No. 3, 2015, pp. 67-74. doi: 10.11648/j.ajnc.20150403.18

Abstract: High performance computing (HPC) encompasses advanced computation over parallel processing, enabling faster execution of highly compute intensive tasks such as climate research, molecular modeling, physical simulations, cryptanalysis, geophysical modeling, automotive and aerospace design, financial modeling, data mining and more. High performance simulations require the most efficient compute platforms. The execution time of a given simulation depends upon many factors, such as the number of CPU/GPU cores and their utilization factor and the interconnect performance, efficiency, and scalability. CPU and GPU clusters are one of the most progressive branches in a field of parallel computing and data processing nowadays. GPUs have become increasingly common in supercomputing, serving as accelerators or "co-processors" in every node CPU-GPU to help CPUs get work done faster. In this paper I use the Multiclass Closed Product-Form Queueing Network (MCPFQN) and Mean Value Analysis (MVA) to analyze effects of the CPU-GPU cluster interconnect on the performance of computer systems.

Keywords: CPU-GPU Clusters, Performance, Multiclass Product Form Queueing Network

1. Introduction

Efficient high performance computing systems require high bandwidth, low latency connections between thousands of multiprocessor nodes, as well as high speed storage.

Parallel computing using accelerators has gained widespread research attention in the past few years. In particular, using GPUs (Graphics Processing Units) for general purpose computing (GPGPU) has brought forth several success stories with respect to time taken, cost, power, and other metrics. . In the most recent list of the world's fastest 500 supercomputers, 53 systems used co-processors and 38 of these used Nvidia chips. The second and sixth most powerful supercomputers used Nvidia chips along side CPUs. Intel still dominates, providing processors for 82.4 percent of Top 500 systems. As the price/performance of GPUs has improved, a number of petaflop supercomputers such as Tianhe-I and Nebulae have started to rely on them. However, other systems such as the K computer continue to use conventional processors such as SPARC-based designs and the overall applicability of GPUs in general purpose high performance computing applications has been the subject of debate, in that while a GPU may be tuned to score well on specific benchmarks its overall applicability to everyday algorithms may be limited unless significant effort is spent to tune the application towards it. China Tianhe-1A, an upgraded supercomputer in 2010, was equipped with every node of 14,336 Xeon X5670 processors and 7,168 NVIDIA Tesla M2050 GPGPUs. It has a theoretical peak performance of 4.701 petaflops. NVIDIA suggests that it would have taken "50,000 CPUs and twice as much floor space to deliver the same performance using CPUs alone." The current heterogeneous system consumes 4.04 megawatts compared to over 12 megawatts had it been built only with CPUs.

The overall most efficient GPU processing model determined thus far is Single Instruction Multiple Data (SIMD). The SIMD model has been leveraged to great advantage in traditional vector processor/supercomputer designs. As displayed in figure 1, the SIMD GPU is nominally organized as an assembly of '*N*' distinct multiprocessors ($N_{MP/TPC}$) in every thread processing cluster (TPC). Each multiprocessor per TPC consists of 'M' cores - distinct thread processors. ($M_{TP/MP}$). There are some TPC in one GPU ($N_{TPC/GPU}$). Total number of thread processors (core) per GPU is:

$$N_{Thread/GPU} = N_{TPC/GPU} N_{MP/TPC} M_{TP/MP} \quad (1)$$

SIMD-Cores share an Instructiuon Unit with other cores in a multiprocessor. Multiprocessors have local registers (splits to local memory), local L1 cache, shared L2 cache, constant cache, texture cache, and shared memory. Constant/texture

cache are readonly and have faster access than shared memory. Global/Device memory for GPU (shared for all multiprocessors) is DRAM (DDR3 or DDR5). Threads are organized into blocks, which are organized into a *grid*. A multiprocessor executes one block at a time. A *warp* is the set of threads executed in parallel. The number of thread per a warp may be same as the number of thread processor cores. Programming model of GPU is heterogeneous Computing, where GPU and CPU execute different types of code. CPU runs the main program, sending tasks to the GPU in the form of kernel functions, and multiple kernel functions may be declared and called, but only one kernel may be called at a time. Nvidia developed Compute Unifed Device Architecture (CUDA), a simple language for GPU computing allows a programmer to use the C programming language to code algorithms for executions on the GPU. Using CUDA or OpenCL programming toolkits many real-world applications can be easily implemented and run significantly faster than on multi-processor or multi-core systems. The CUDA framework allows to lower the CPU load and send the computations to the GPU which by construction has a highly parallel architecture, thus accelerating the processing of large amounts of data arrays using an identical set instructions (called kernel) for each element in one data array. In this way the processor is only busy with supplying the data and the kernels to the GPU and then collecting the results. Although the transfer between CPU (host) and GPU (device) might appear as a bottleneck due to the slow speed of the bridge bus. One could obtain very good results if it exposes as much data parallelism as possible in the algorithm and also takes care of mapping it to the hardware as eficiently as possible in such a way that the transfers are minimized.

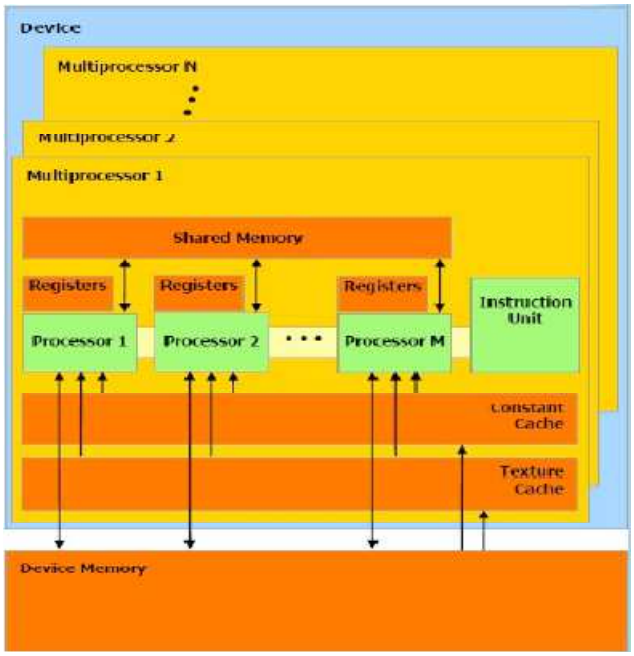

Figure 1. *GPU hardware.*

2. Choice of the CPU-GPU Cluster

Typical cluster consists of homogenous Central Processing Units (CPUs). A new model for parallel computing based on using CPUs and GPUs together to perform a general purpose scientific and engineering computing was developed in the last years, and used to solve complex scientific and engineering problems. Craving for more computational power led to the idea of plunging the CUDA framework in a Message Passing Interface (MPI) environment, thus arising the concept of cluser of GPUs. Inside the cluster each node of the MPI network would pass most of intensive parallel tasks to the GPU, off-loading the CPU which is free to handle the network communication between nodes. A cluster is a computer system comprising two or more computers ("nodes") connected with a high-speed network. Cluster computers can achieve higher availability, reliability, and scalability than is possible with an individual computer. There are three principal components used in a GPU cluster: host nodes, GPUs, and interconnect.

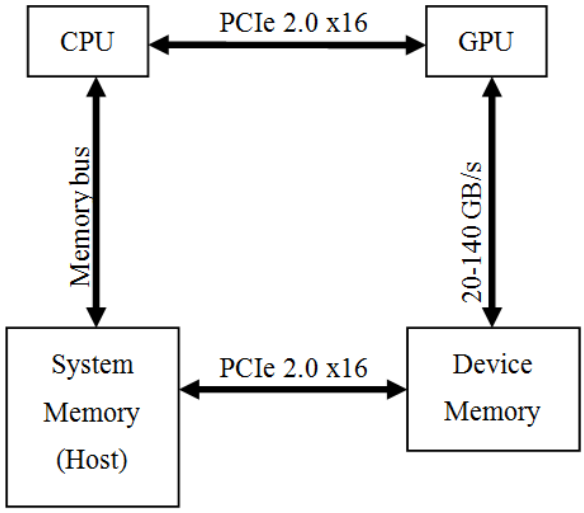

Figure 2. *CPU/GPU interconnect.*

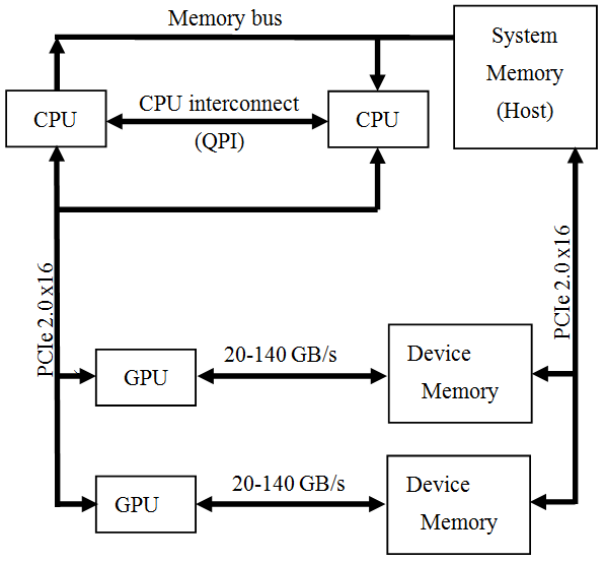

Figure 3. *2CPU/2GPU interconnect.*

Since the expectation is for the GPUs to carry out a substantial portion of the calculations, host memory, PCIe bus,

and network interconnect performance characteristics need to be matched with the GPU performance in order to maintain a well-balanced system. In particular, high-end GPUs, such as the NVIDIA Tesla, require full-bandwidth PCIe Gen 2 x16 slots that do not degrade to x8 speeds when multiple GPUs are used. Also, InfiniBand QDR interconnect is highly desirable to match the GPU-to-host bandwidth. Host memory also needs to at least match the amount of memory on the GPUs in order to enable their full utilization, and a one-to-one ratio of CPU cores to GPUs may be desirable from the software development perspective as it greatly simplifies the development of MPI-based applications.

The figures 2 and 3 show typical CPU/GPU interconnect with 1 and 2 CPUs and 2GPU. The figure 4 shows a CPU/GPU desktop supercomputer organized in clusters.

The System interconnect (cluster interconnect) is 1GbE/10GbE Ethernet or InfiniBand switch. Every node in a cluster consist of one/or two multicore-multithreading CPU chips on two sockets with high speed CPU interconnect, e.g. used Intel QuickPath Interconnect – QPI with a clock rate of 3.2 GHz yields a data rate of 25.6 GB/s = (3.2 GHz x 2 bits/Hz (double data rate) x 20 (QPI link width) x (64/80) (data bits/flit bits) x 2 (unidirectional send and receive operating simultaneously))/8 (bits/byte), or AMD HyperTransport bus with a clock rate 3.2 GHz yields a data rate of 25.6 GB/s = (3.2 GHz x 2 bits/Hz (double data rate) x 32 bits/s))/8 bits/byte; two/or three multiprocessor-multithreading GPU installed on PCI Express 2 x16 slots (for all sorts of graphics cards, can access a total bandwidth of 8 GB/s). The Cluster Interconnect is QDR (40 Gbps) InfiniBand connectivity on each node in one cluster. An Infiniband link is a serial link operating at one of five data rates: single data rate (SDR) switch chips have a latency of 200 ns (12X: 24 Gbit/s), double date rate (DDR) switch chips have a latency of 140 ns (12X: 48 Gbit/s and quad data rate (QDR) switch chips have a latency of 100 ns (12X: 96 Gbit/s), fourteen data rate (FDR-10, FDR) (12X: 163.64 Gbit/s), and enhanced data rated (EDR) (12X: 300 Gbit/s). Larger systems with 12X links are typically used for cluster and supercomputer interconnects and for inter-switch connections. For cluster interconnect, the InfiniBand uses a switched fabric topology, as opposed to a hierarchical switched network like traditional Ethernet architectures, although emerging Ethernet fabric architectures propose many benefits which could see Ethernet replace InfiniBand. Most of the cluster interconnect topologies are Fat-Tree, 2D-mesh/2D-Torus or 3D-Torus, butterfly (Clos) as well.

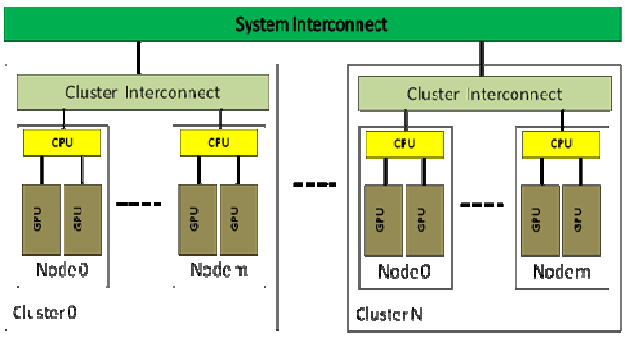

Figure 4. *CPU/GPU supercomputer organized in clusters.*

In this context, multiprocessor operation is defined modulo an ensemble of threads scheduled (by hardware scheduler) and managed as a single entity across thread processing clusters. In this manner, shared-memory access, SIMD instruction fetch and execution, and cache operations are maximally synchronized. Memory is organized hierarchically *Global -> Device -> Shared* where Global/Device memory transactions are understood as mediated by high-speed bus transactions (PCIe, HyperTransport, or QPI). A key subtlety associated with the CPU/GPU processing architecture is GPU processing is effectively *non-blocking*. Thus, CPU processing may continue as soon as a work-unit has been written to the GPU transaction buffer. Host (CPU) processing and GPU processing may be overlapped as displayed in figure 1. In principle, GPU work unit assembly/disassembly and I/O at the GPU transaction buffer may to large extent be hidden. In such case, GPU performance will effectively dominate system performance. As might be expected, optimal GPU processing gain is achieved at an I/O constraint boundary whereby thread processors never stall due to lack of data.

At an application level, the maximum achievable speedup is governed by Amdahl's Law; any acceleration (*'A'*) due to thread parallelization will critically depend upon: (1) the fraction of code than can be parallelized (*'P'*), (2) the degree of parallelization (*'N'*), and (3) any overhead associated with parallelization. Thus, expected acceleration is modeled by:

$$A = \frac{1}{(1-P)+P/N} \tag{2}$$

A key consideration is the limiting case:

$$\lim_{N\to\infty}\left(\frac{1}{(1-P)+P/N} = \frac{1}{1-P}\right) \tag{3}$$

This indicates a theoretical maximum acceleration for the complete application. However, CPU code pipelining, (i.e. overlap with GPU processing), must also be factored into any calculation for '*P*'; pipelining effectively parallelizes CPU and GPU code segments reducing the non-parallelized code fraction (1-*P*). Thus, under circumstances where decrease is sufficient to claim $\frac{P}{N} >> (1-P)$, Amdahl's Law then becomes:

$$A \cong \frac{1}{\frac{P}{N}} = \frac{N}{P} \cong N \tag{4}$$

CPU/GPU-based desktop supercomputing model is cluster architecture (figure 1), that characteristic acceleration values approaching a limit:

$$A_{Cluster} \cong N_{Node/CL} N_{Thread/GPU} N_{GPU} \tag{5}$$

$N_{Node/CL}$ - number of nodes per cluster,

N_{GPU} - number of GPUs in the cluster .

Total acceleration of the system with N_{CL} clusters:

$$A_{Supercomputer} \cong (N_{Node} N_{Thread/GPU} N_{GPU}) N_{CL} \quad (6)$$

3. The MCPFQN of the CPU/GPU Node

Queueing networks [1] consisting of several service stations are more suitable for representing the structure of many systems with a large number of resources than models consisting of a single service station. In a queueing network (QN) at least two service stations are connected to each other. A station, i.e., a node, in the network represents a resource in the real system. Jobs in principle can be transferred between any two nodes of the network; in particular, a job can be directly returned to the node it has just left. The QN is called open when jobs can enter the network from outside and jobs can also leave the network. Jobs can arrive from outside the network at every node and depart from the network from any node. The QN is said to be closed when jobs can neither enter nor leave the network. The number of jobs in a closed network is constant. Jobs can be different in their service times and in their routing probabilities. Jobs with same service times and routing probabilities belong to one class. So QNs can be single class or multiclass networks (MCQN). It is also possible that a job changes its class when it moves from one node to another. If the QN contains both open and closed classes, then it is said to be the mixed network. Behaviours of many queueing system models can be described using Continuous –time Markov chains (CTMCs). The QNs that have an unambiguous solution of the local balance equations are called product-form queueing networks (PFQNs). The term product-form of open and closed queueing networks with exponentially distributed interarrival and service times. The queueing discipline at all stations was assumed to be FCFS. As the most important result for the queueing theory, it is shown that for these networks the solution for the steady-state probabilities can be expressed as a product of factors describing the state of each node. This solution is called product-form solution. For the performance analysis of CPU/GPU architectures, we can use MCPFQN with fixed number of jobs, which are threads.

Figure 5 shows the p_{MC21} MCPFQN model of the CPU/GPU architecture in figure 3. 2 CPUs are 2-core multithreading: CPU 1 server node (include private L1 and L2 caches) with: CPU core 11 and CPU core 21, CPU 2 server node: CPU core 21 and CPU core 22; every threads are modeled by thread processor with service rate μ_T. CPUint node with service rate μ_{int} is the CPU interconnect (e.g. Intel QPI or AMD HyperTransport bus). 2 GPUs (GPU 1 and GPU 2 nodes include caches, have local thread processors with service rate μ_{TP}) are installed on the PCIe 2.0 x 16 slots (node CPU-GPU int with service rate μ_{PCi}) of CPU mainboard. The Memory bus (via MCH chip of the chipset) is the node Memory bus with service rate μ_{MB}. Memory modules are nodes System Memory with service rate μ_M. Every GPU connect to the local Device Memory modules (with service rate μ_{DM}) via interconnect bus (nodes GPU1-Device Memory, GPU2-Device Memory) with service rate μ_{GM}.

The connection between System memory and Device Memories may be PCIe 2.0 x16 or Infiniband (node DM-SM int with service rate μ_{MG}). Nodes CPU and GPU are type M/G/m-PS with service disciplines: FCFS, all other nodes are type M/M/m – FCFS.

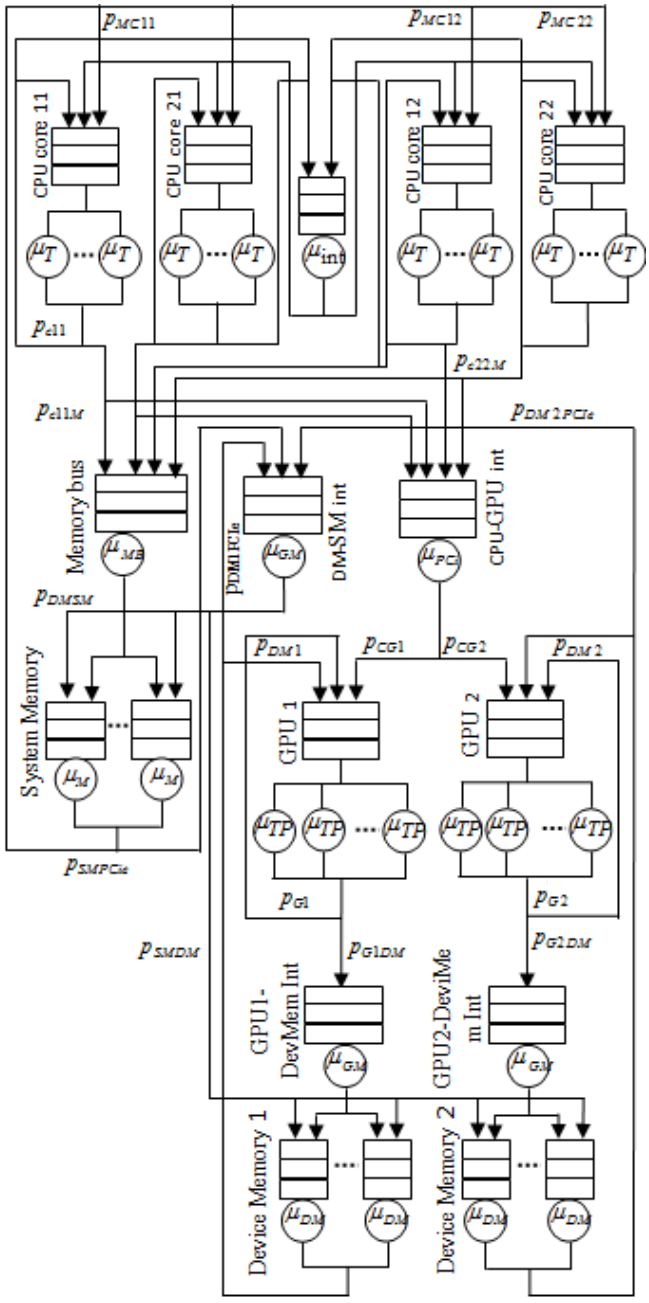

Figure 5. *Closed queue network of 2CPU/2GPU desktop supercomputer model in figure 3.*

Assume that: GPUs and CPUs (include private L1 and L2 caches) work in 2GHz have average service time for every tread is 0.5ns $(\mu_T = \mu_{TP} = 0.5ns)$. From every CPU there are 4 routing directions with probabilities (their sum = 1): core loopback $(p_{c11}, p_{c21}, p_{c12}, p_{c22})$, to other core in one CPU (p_{c11c21}, p_{c12c22}), to other CPU

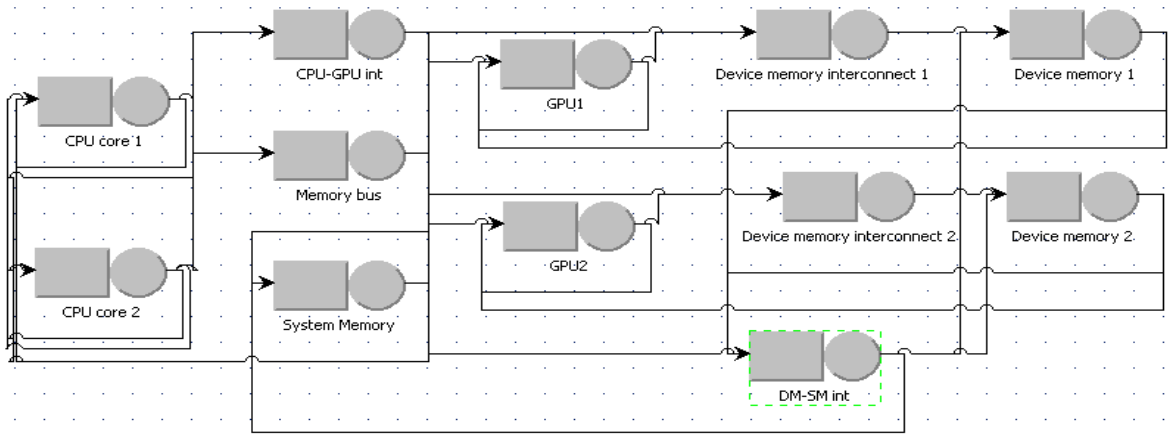

Figure 6. *MCPFQN of the CPU-2-core-4-thread/2GPU-448-core desktop architecture.* $(p_{c11\text{int}}, p_{c21\text{int}}, p_{c12\text{int}}, p_{c22\text{int}})$ *and to System memory* $(p_{c11M}, p_{c21M}, p_{c12M}, p_{c22M})$.

From every GPU there are 2 routing flows: GPU loopback (p_{G1}, p_{G2}) and to Device memories (p_{G1DM}, p_{G2DM}).

From CPUs to GPUs (via node CPU-GPU int) there are 2 flows: to GPU1 (p_{CG1}), to GPU2 (p_{CG2}).

From GPU Device memories there are 2 routing flows: loopback to GPU (p_{DM1}, p_{DM2}) to bus connect (node DM-SM int) to System memory $(p_{DM1PCIe}, p_{DM2PCIe})$.

From node DM-SM int there are 2 routing flows: to System memory (p_{DMSM}), to Device memory (p_{SMDM}).

From System memory there are 2 routing flows: to CPU cores $(p_{MC11}, p_{Mc12}, p_{MC21}, p_{MC22})$.

For QN, most important performance measures are:

- Marginal probabilities $\pi_i(k)$ for closed queueing networks, that ith node in the state $S_i = k$, is :

$$\pi_i(k) = \sum_{\substack{\sum_{j=1}^{N} S_j = K \\ \& S_i = k}} \pi(k_1, ..., k_N) \tag{7}$$

- Utilization ρ_{ir} of the *i*th node with respect to jobs of the *r*th class is:

$$\rho_{ir} = \frac{1}{m_i} \sum_{\substack{\text{all states } k \\ \text{with } k_r > 0}} \pi_i(k) \frac{k_{ir}}{k_i} \min(m_i, k_i),\ k_i = \sum_{r=1}^{R} k_{ir} \tag{8}$$

And if the service rates are independent on the load:

$$\rho_{ir} = \frac{\lambda_{ir}}{m_i \mu_{ir}}. \tag{9}$$

Throughput λ_{ir} is the rate at which jobs of the *r*th class are services and leave the *i*th node:

$$\lambda_{ir} = \sum_{\substack{\text{all states } k \\ \text{with } k_r > 0}} \pi_i(k) \frac{k_{ir}}{k_i} \mu_i(k_i) \tag{10}$$

Or if the service rates are independent on the load: $\lambda_{ir} = m_i . \rho_{ir} . \mu_{ir}$.

- System throughput λ_r of jobs of the *r*th class:

$$\lambda_r = \frac{\lambda_{ir}}{e_{ir}} \tag{11}$$

Mean number of Jobs (or customer number) $\overline{K}_{ir}$ of the rth class at the *i*th node is:

$$\overline{K}_{ir} = \sum_{\substack{\text{all states } k \\ \text{with } k_r > 0}} k_r . \pi_i(k) \tag{12}$$

Little's theorem can also be used here: $\overline{K}_{ir} = \lambda_{ir} . \overline{T}_{ir}$

Mean Response Time $\overline{T}_{ir}$ of the jobs of the rth class at the ith can also be determined using Little's theorem:

$$\overline{T}_{ir} = \frac{\overline{K}_{ir}}{\lambda_{ir}} \tag{13}$$

- Mean Waiting Time $\overline{W}_{ir}$: if the service rates are load-independent, then the mean waiting time is given:

$$\overline{W}_{ir} = \overline{T}_{ir} - \frac{1}{\mu_{ir}} \tag{14}$$

For analysis, performance parameters are: Number of customers, Response Time, Utilization, Throughput, and System Throughput in relation with service times at interconnect networks (e.g. PCIe 2.0 x16) between CPU and GPU (node CPU-GPU int), and between System memory and Device memories (node DM-SM int). The analysis is made for architectures: 2-core/4-threading CPU, 2 GPUs with 448 thread cores in each, number of jobs: 448, and exponential distribution service time, $f(t) = \lambda e^{-\lambda t}$ of interconnects (CPU-GPU int) and DM-SM int in figure 6) for three cases: i)

mean: 0.25, $\lambda = 4$; ii) mean: 1, $\lambda = 1$; iii) mean: 4, $\lambda = 0.25$. Results of Performance parameters are given in figures 7. For the performane analysis, I take only response times and system throughput of three cases.

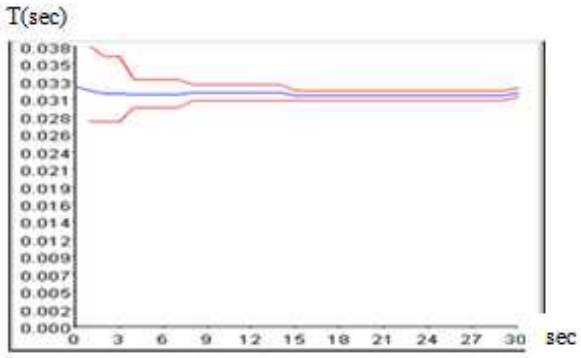

Figure 7(a). *System Response Time T (seconds): mean: 4167.3293, min:4047.4249, max:4287.2338 in case i) Secvice Time distribution of CPU-CPU int=DM-SM int: mean:0.25.* $\lambda = 4$.

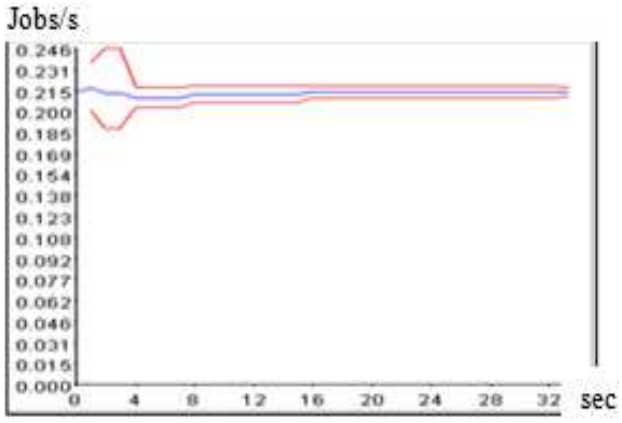

Figure 7(b). *System Throughput (jobs/sec): mean: 0.2143, min:0.2108, max:0.2178 in case i) Secvice Time distribution of CPU-CPU int=DM-SM int: mean:0.25.* $\lambda = 4$.

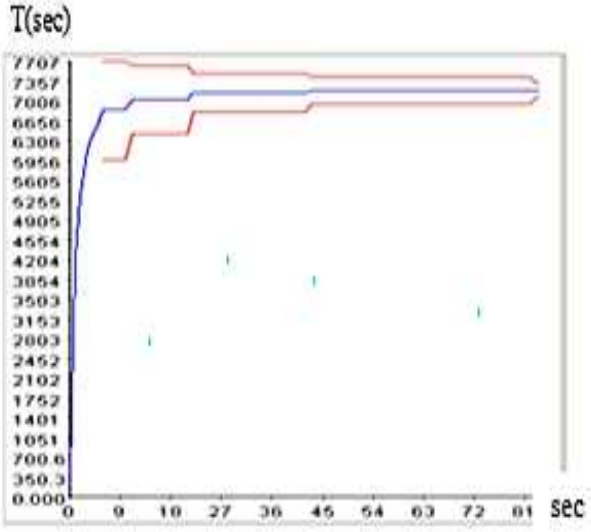

Figure 7(c). *System Response Time T (seconds): mean: 7188.6417, min:7073.0428, max:7304.2406 in case i) Secvice Time distribution of CPU-CPU int=DM-SM int: mean:1.* $\lambda = 1$.

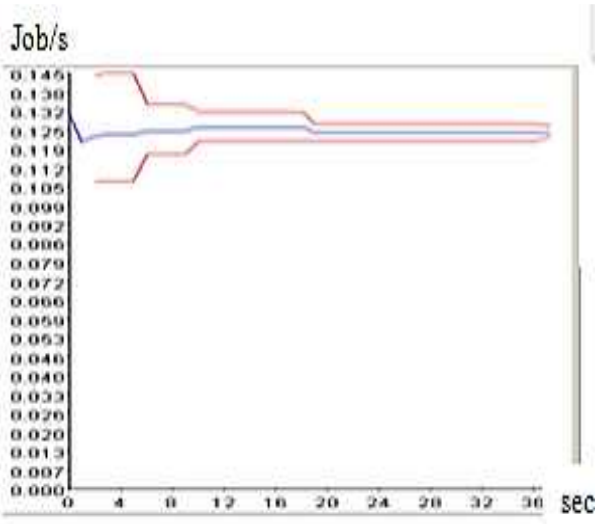

Figure 7(d). *System Throughput (jobs/sec): mean: 0.1244, min:0.1223, max:0.1265 in case i) Secvice Time distribution of CPU-CPU int=DM-SM int: mean:1,* $\lambda = 1$.

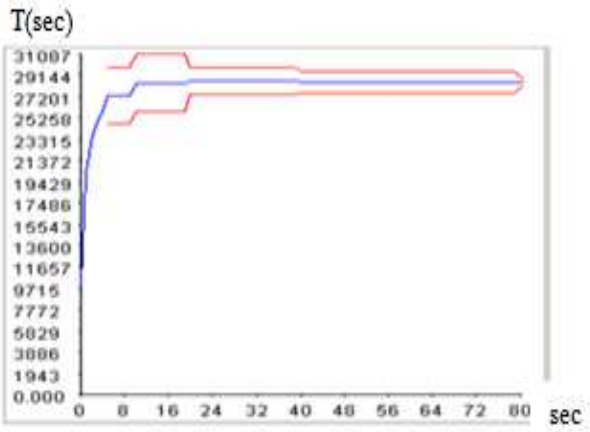

Figure 7(e). *System Response Time: mean: 2.853E4, min:2.807E4, max:2.900E4 in case iii) Secvice Time CPU-CPU int=DM-SM int: mean:4,* $\lambda = 0.25$

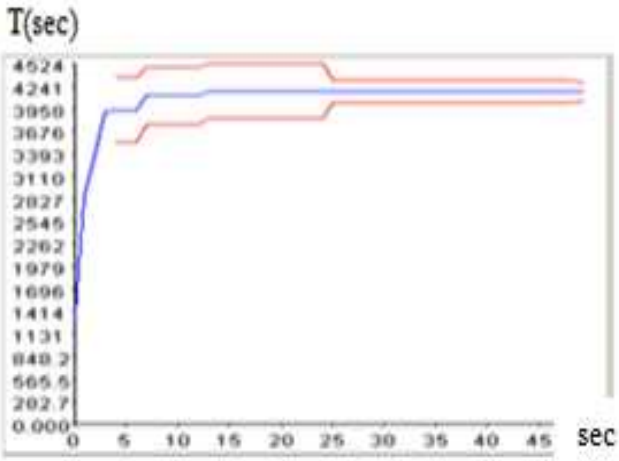

Figure 7(f). *System Throughput: mean: 0.0313, min:0.0307, max:0.0320 in case iii) Secvice Time distribution of CPU-CPU int=DM-SM int: mean:4,* $\lambda = 0.25$

Table 1. *Avarage service times of queue nodes.*

	Class 1	Class 2	Class 3	Class 4	Class 5
CPU core 1	1.0	0.5	0.25	0.15	0.1
CPU core 2	1.0	0.5	0.25	0.15	0.1
GPU 1	1.0	0.5	0.25	0.15	0.1
GPU 2	1.0	0.5	0.25	0.15	0.1
CPU-GPU int	4.0	2.0	1.0	0.5	0.25
GPU1-DevMem	2.0	1.0	0.5	0.25	0.2
GPU2-DevMem	2.0	1.0	0.5	0.25	0.2
Device memory 1	40.0	35.0	30.0	25.0	20.0
Device memory 2	40.0	35.0	30.0	25.0	20.0
Memory bus	2.0	1.0	0.5	0.25	0.2
System memory	40.0	35.0	30.0	25.0	20.0
DM-SM int	4.0	2.0	1.0	0.5	0.25

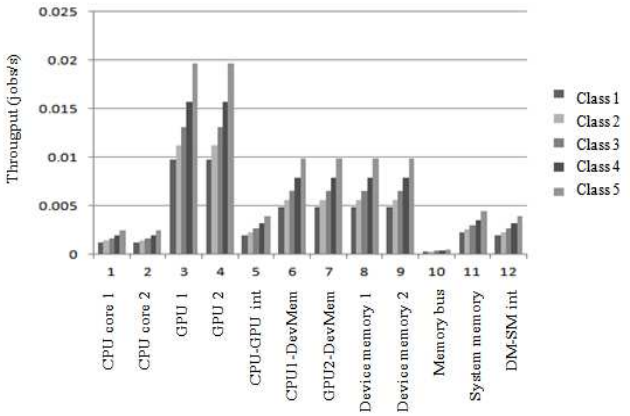

Figure 8. *Average service times for the MVA algorithm.*

Table 2. *Throughput of the CPU-GPU node.*

	Class 1	Class 2	Class 3	Class 4	Class 5
CPU core 1	0.0012	0.0014	0.0016	0.0020	0.0025
CPU core 2	0.0012	0.0014	0.0016	0.0020	0.0025
GPU 1	0.0098	0.0112	0.0131	0.0158	0.0197
GPU 2	0.0098	0.0112	0.0131	0.0158	0.0197
CPU-GPU int	0.0020	0.0023	0.0026	0.0032	0.0039
GPU1-DevMem	0.0049	0.0056	0.0066	0.0079	0.0098
GPU2-DevMem	0.0049	0.0056	0.0066	0.0079	0.0098
Device memory 1	0.0049	0.0056	0.0066	0.0079	0.0098
Device memory 2	0.0049	0.0056	0.0066	0.0079	0.0098
Memory bus	2.5E-4	2.8E-4	3.3E-4	4.0E-4	4.9E-4
System memory	0.00227	0.0026	0.0030	0.0035	0.0044
DM-SM int	0.00201	0.0023	0.0026	0.0032	0.0039

Based on the MCPFQN in the figure 6, we can use Mean Analysis (MVA) algorithm [1] for calculating performance measures. For this case, I define CPU-GPU node: CPU core 1 and CPU core 2 by 6-core, GPU 1 and GPU 2 with 512 cores. Average service times (ns) of all nodes and five classes are listed in table 1 and in figure 8. The throughput (jobs/s) of the CPU-GPU node with the initial setting 12 jobs (number of customers) in CPU core 1 and CPU core 2 nodes for all classes are listed in table 2 and in figure 9.

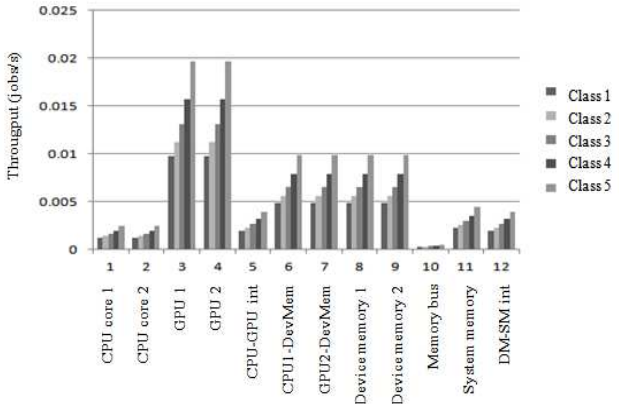

Figure 9. *Throughput of CPU-GPU node is taken by the MVA algorithm.*

4. Conclusion

The performance rerults in figures 7 and figure 9 of MCPFQN model of the CPU-GPU node: system response times T (sec) and system throughput (jobs/s) show that: The changes of interconnect service times affect deeply system performance parameters: system response time and throughput that belong to service times of CPU-GPU interconnect and GPU device memory-CPU system memory interconnect, in fixed average service rates of all other nodes. Clearly, more service time of interconnects, more system response time and less system throughput. Using MCPFQN, we can evaluate system performance depend on any other node or complet CPU+GPU desktop supercomputers with many cluster and diffrent cluster-to-cluster interconnect network topologies and delays.

References

[1] Gunter Bolch, Stefan Greiner, Hermann de Meer, Kishor S.Trivedi: Queueing Networks and Markov Chains; A John Wiley & sons, Inc., Publication.

[2] Peng Wang, NVIDIA. Fundamental Optimizations in CUDA, GPU technology conference.

[3] Bryan Schauer. “Multicore Processors-A Necessity”, 9/2008.

[4] John Mellor-Crummey, Department of Computer Science Rice University: Caching for Chip Multiprocessor, 8/2009.

[5] Sarah Bird, …University of Texas at Austin, IBM Austin: Performance Characterization of SPEC CPU Benchmarks on Intel’s Core Microarchitecture based processor. 2006.

[6] R.Ubal, J.Sahuquillo,..:Multi2Sim. A Simulation Framework to Evaluate Multicore-Multithreaded Processors, 2006.

[7] W.M.Zuberek. Performance eqivalence in the simulation of Multiprocessor systems, 2002.

[8] Scott.T.Lentenegger, Mary K.Vernon. A mean-Value performance Analysis of a New Multiprocessor Architecture, 12/1988.

[9] Angel Vassilev Nikolov, National University of Lesotho, 180, Roma, May,2009. Model of a shared Memory Multiprocessor.

[10] Susmit Biswas, Diana Franklin,,,2008. Multi-Execution. Multicore Caching for Data- Similar Executions.

[11] Intel Multicore microprocessor technology. http://www.Intel.ccom/.

[12] Rafael H. Saavedra-Barrera, David E.Culler. An analytical solution for a markov chain modelling multithreaded execution.

[13] Michael R.Marty, University of Wisconsin-madison, 2008. Cache coherence techniques for multicore processors.

[14] Richard Mcdougall and James Laudon. Multi-core microprocessors are here.

[15] Avinatan Hassidim, 16/09/2009. Cache replacement Policies for Multicore processors.