HAL
open science

mHLogGP: A Parallel Computation Model for CPU/GPU Heterogeneous Computing Cluster

Gangfeng Liu, Yunlan Wang, Tianhai Zhao, Jianhua Gu, Dongyang Li

To cite this version:

Gangfeng Liu, Yunlan Wang, Tianhai Zhao, Jianhua Gu, Dongyang Li. mHLogGP: A Parallel Computation Model for CPU/GPU Heterogeneous Computing Cluster. 9th International Conference on Network and Parallel Computing (NPC), Sep 2012, Gwangju, South Korea. pp.217-224, 10.1007/978-3-642-35606-3_25 . hal-01551314

HAL Id: hal-01551314

https://inria.hal.science/hal-01551314v1

Submitted on 30 Jun 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

mHLogGP: a Parallel Computation Model for CPU/GPU Heterogeneous Computing Cluster

Gangfeng Liu, Yunlan Wang, Tianhai Zhao, Jianhua Gu, Dongyang Li

Center for High Performance Computing, Northwestern Polytechnical University, Xi'an, China

wangyl@nwpu.edu.cn

Abstract. CPU/GPU heterogeneous computing has become a tendency in scientific and engineering computing. The conventional computation models cannot be used to estimate the application running time under the CPU/GPU heterogeneous computing environment. In this paper, a new model named mHLogGP is presented on the basis of mPlogP, LogGP and LogP. In mHLogGP, the communication and memory access is abstracted based on the characteristic of CPU/GPU hybrid computing cluster. This model can be used to study the behavior of application, estimate the execution time and guide the optimization of parallel programs. The results show that the predicted running time approaches to the actual execution of program.

Keywords: CPU/GPU Heterogeneous computing; Parallel Computation Model; Parallel Computing

1 Introduction

The compute-capable graphics processing units (GPUs) are attractive as accelerators for high-performance parallel computing. According to the top500 supercomputer sites, three out of the top five fastest supercomputers in November 2011, employ GPUs. CPU and GPU have their own advantages respectively. CPU is suitable for complex arithmetical, logical, and input/output operations. GPU consists of hundreds of smaller cores which gives high compute performance. Therefore, heterogeneous CPU/GPU computing provides new and powerful computing architecture.

The conventional parallel computation models are not suitable for the new emerging heterogeneous computing environment. The PRAM [1] is unrealistic because it assumes that all processors work synchronously. The bulk-synchronous parallel model (BSP) [2] models asynchronously processors, communication latency and limited bandwidth. The LogP model [3] is capable of predicting the communication time for short messages but cannot accurately predict the communication time for long messages. The LogGP [4] extends it and adds an additional parameter for long messages, which captures the bandwidth constraints for long messages. The PlogP [5] considers the varying communication overheads for different message sizes. The memory LogP [6], $\log_3$P and $\log_n$P [7] models draw characteristics of data movement in the tradi-

tional parallel computer. However, all the models mentioned above ignore the new characteristics of communication and memory access in the heterogeneous computing system. MMGP [8] is a multi-dimensional parallel computation model for steering the parallelization process on heterogeneous multi-core processors. However, it ignores the influence of data movement across computational cores. mPlogP [9] is also a model for heterogeneous multi-core computer. It extends PLogP and adds another level on top of the conventional level to model behaviors of computational cores. However, the characteristics of CPU/GPU hybrid computing environment is not considered in this model.

In this paper, a new model named mHLogGP is proposed, which is on the basis of mPlogP，LogGP and LogP. In order to analyze the characteristics of CPU/GPU heterogeneous computing cluster, some related parameters are defined to estimate the time performance of the parallel program.

2 mHLogGP model

2.1 abstract hardware architecture of GPU and CPU

Fig.1(a) shows GPU abstract architecture [10]. Each thread processors has private memory and parallel data cache. The thread processors visit each other through the parallel data cache and global memory. Fig.1 (b) shows CPU abstract architecture, the CPU cores exchange message via the bus and the host memory.

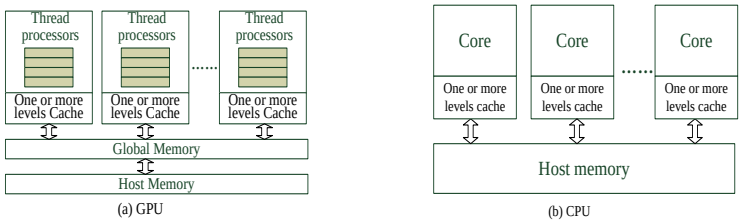


Fig. 1. Hardware Abstraction of Processor Architecture

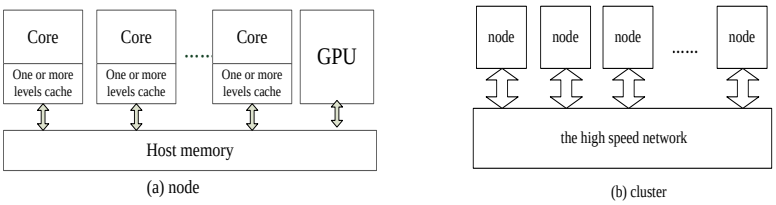


Fig. 2. Hardware Abstraction of a node(a) and cluster(b)

The hardware abstraction of computer node which includes GPU and CPU is showed in Fig.2(a). The host memory is shared by GPU and CPU. GPU exchanges information with CPU by accessing host memory.

The hardware abstraction of the widely used CPU/GPU computer cluster is shown in Fig.2(b). A computer cluster consists of dozens to thousands computers that are connected by high performance network such as Infiniband, 10-Gigabit Ethernet, etc.

2.2 The parameters of the model

o(Overhead): the time needed by processor to send or receive a message, $o \in \{o_s, o_r\}$. $o_s \in \{o_{sg}, o_{sc}\}$, o_{sg} is the time that GPU sends messages to host memory. o_{sc} is the time that CPU sending messages. $o_{sc} \in \{o_{scd}, o_{scn}\}$, o_{scd} defined as the time that CPU sends messages to global memory. o_{scn} is the time that CPU sending messages to another computer node. $o_r \in \{ o_{rgb}, o_{rn}, o_{rd} \}$, o_r is the time that a processor receive a message. o_{rgb} is the time of initializing the GPU. o_{rn} (o_{rd}) is defined as the time that CPU (GPU) engaged reception of each message.

l : latency, or delay, incurred in communicating a message from its sender to receiver. $l \in \{ l_g, l_c, l_{dm}, l_{nn}\}$, l_g denotes the latency between the parallel data cache and the global memory. l_c denotes the latency from the core of CPU to the host memory of another node. l_{dm} denotes the latency between the global memory and the host memory. l_{nn} is defined as the delay from one node to another.

G: the Gap per byte for long messages, which is defined as the time per byte for a long message. The reciprocal of *G* characterizes the available communication bandwidth per processor for long messages.

g: the gap, defined as the time interval between consecutive message transmission or reception. The reciprocal of g corresponds to the available communication bandwidth per-processor and the speed of accessing the memory. $g \in \{ g_g, g_c,\}$, g_g means the time interval between consecutive message transmission or reception in the global memory of GPU. g_c denotes the time interval for sending a message from the host memory to network cache.

P: the number of computer nodes. P is mostly used to model the collective communication.

3 Usage of the Model

3.1 Point-to-Point Communication

In this section, the performance of point-to-point communications is predicted. In the CPU/GPU hybrid computing environment, the cost of sending short message between two nodes can be predicted as $o_{s1} + l + o_{r1}$(LogP), where *l* is the latency of communicating a message from its sender to receiver, o_{s1}, o_{r1} is the time of sending or receiving the first message, where G is the Gap per byte. For long message, the Point-to-Point Communication time can be predicted as follows:

$$T = o_{s1} + (k_1 - 1)G + l + o_{r1} \quad \text{(LogGP)} \tag{1}$$

As shown in Fig.3, if the non-blocking communication is used to send ‘m’ consecutive messages, the transmission time is as follows:

$$T = o_{s1} + (k_1 - 1)G + \sum_{i=1}^{m-1} \max(((k_{i+1} - 1)G + g_{c(i+1)} + o_{s(i+1)}), l_{nni}) + l_{nnm} + o_{rm} \quad (2)$$

Where k_{i+1} is the size of the (i+1)-th message. o_{si}, o_{ri} is the time of sending or receiving the i-th message, It is obvious that network bandwidth and the speed of accessing the memory have a great influence on inter-node communication time.

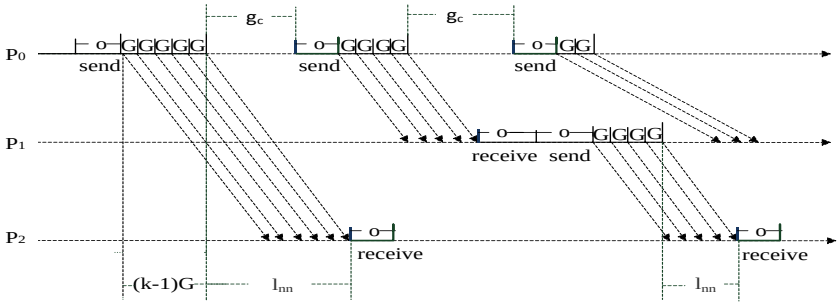


Fig. 3. Sending and receiving messages

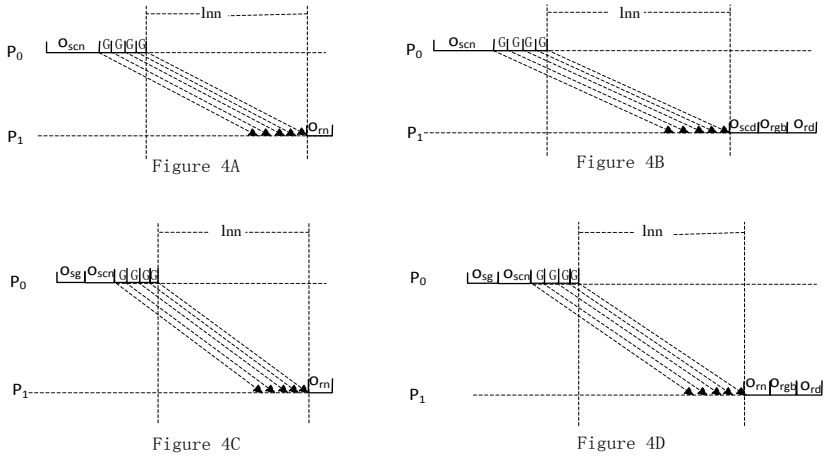


Fig. 4. Half-round trip sender/receiver communication cost

The communication cost varies with different sender and receiver. If messages are send from CPU of one node to CPU of another node (Fig.4A), the communication cost can be expressed by (2) where $o_s = o_{scn}$ and $o_r = o_{rn}$. If messages are send from CPU of one node to GPU of another node (Fig.4B), $o_s = o_{scn}$, for the first message, $o_r = o_{rgb} + o_{rn} + o_{rd}$, for other messages, $o_r = o_{rn} + o_{rd}$. If messages are sent from GPU of one node to CPU of another node (Fig.4C), $o_s = o_{sg} + o_{scn}, o_r = o_{rn}$. If messages are send from GPU of one node to GPU of another node (Fig.4D), $o_s = o_{sg} + o_{scn}$, for the first message $o_r = o_{rgb} + o_{rn} + o_{rd}$, for other messages $o_r = o_{rn} + o_{rd}$.

3.2 Broadcast Communication

We take the MPICH2 as example. For the short message or the number of nodes less than 8, the broadcast communication is taken by the short message algorithm, namely, binomial tree. If the number of nodes is P and the size of the message is k. it needs lgP steps in broadcast communication. Each byte travels through the network for l_{nn}. Under the CPU/GPU heterogeneous computing environment, the broadcast communication cost can be predicted as follows:

$$T_{broP} = \lg P * o + \lg P * k * (G + l_{nn}). \quad (3)$$

For medium-size messages and power-of-two number of nodes, the broadcast communication uses a recursive doubling algorithm. This costs approximately $T_{broP} = 2*\lg P * o + 2*k*(P-1)/P*(G+l_{nn})$. For long messages or medium-size messages and non-power-of-two node situations the broadcast communication use a ring algorithm, which takes P-1 steps, so the total cost is $T_{broP} = (\lg P + P - 1)*o + 2*k*(P-1)/P*(G+l_{nn})$.

4 Validating the Model

Our experimental platform is the Dawning TC3600 Blade system with Intel X5650 processors and NVIDIA Tesla C2050 GPUs at the National Super Computer Center in Shenzhen, China. The computing network is 40Gb InfiniBand.

Table 1. Sender and receiver overheads between Global memory and host memory

Message Size	1B	4KB	16KB	64KB	256KB	1MB	4MB	16MB	256MB
o_{scd},o_{rd}(μs)	3	45	52	64	173	381	1243	4754	71799
O_{rg}(μs)	20	22	26	50	174	824	1329	4803	70209
O_{scn}(μs)	4	4	8	17	32	65	128	904	20426
O_{rgb}(μs)	102055	103537	109984	109455	114427	111911	101656	101410	103542
o_{rn}(μs)	3	5	8	15	32	67	129	1002	21527
l_{nn}(μs)	141	149	225	268	339	755	2589	10461	161469

It can be shown in Table 1 that o_{scd} remains unchanged when the message size is less than 256 KB. It increases linearly when the message size is larger than 256 KB. So does o_{rg}, o_{scn} and o_{rn} when the size is larger than 64KB. Since the GPU must be initialized, no matter the size of message short or large, the value of o_{rgb} has no large variation. Because the speed of intra-node communication is very high, the value of l_g, l_c and l_{dm} can be ignored. l_{nn} denotes the intra-node delay. The experiments show that l_{nn} is random from 100 μs to 700 μs when the size of a message is less than 1MB. It increases linearly when the message size is larger than the 1MB.

As shown in Table 2, G is not stable when the size of a message is less than 1MB. When the message size larger than 1MB, it approaches a constant. g_g and g_c has changed little when the size of the data varies.

As shown in Table 3, for the messages size larger than 64KB, the total cost of broadcast communication is accord with ring algorithm. There may be errors of the

test function, and other factors deviating from the principle of broadcast communication when the messages size less than 64KB.

Table 2. Overheads per byte for a long message

Message Size	**1B**	**256B**	**1KB**	**16 kB**	**64kB**	**256kB**	**1MB**	**16MB**	**64MB**
G(us/B)	141	0.578125	0.145508	0.014710	0.004577	0.001537	0.000845	0.000737	0.000749
g_g(μs)	4	4	4	4	4	7	68	104	249
g_c(μs)	0	0	0	42	39	40	41	55	35

Table 3. Overheads of broadcast communication

size / time(us) / nodes	**1B**	**256B**	**16KB**	**64KB**	**256kB**	**1MB**	**4MB**	**16MB**	**64MB**	**256MB**
2	83667	85284	97732	106578	106578	95334	101449	140239	211425	417945
4	132472	142002	140794	188769	213175	197967	234129	284768	392290	805469
8	207340	203330	291777	294295	299879	300807	472381	599026	822744	1804826

5 Experiments Validation

The matrix multiply [11] is adopted to validate mHLogGP. The implementation of algorithm mainly adopts the point-to-point communication and broadcast communication. Let A and B be two matrices, n represents the number of lines in matrices A. Matrices A is divided into p parts. The master process sends every part and broadcast matrices B to p slave processes. The slave processes receive the data and do the calculation by GPU and send the results to the master process. The master process receives the results and save the information. The overhead of program can be estimated based on the formulas in Section 3.

Table 4. The comparison between computing cost and total cost

Matrix size	**64*64**	**128*128**	**256*256**	**512*512**	**1024*1024**	**2048*2048**	**4192*4192**	**8192*8192**
Total cost(μs)	110268	109819	115474	115553	119466	219825	964722	6999979
Computing cost(μs)	164	166	241	300	423	1042	2854	11284

The cost of sending messages (for the master process) is:

$$T_{ptop1} = o_{scn1} + (k_1 - 1)G + \sum_{i=1}^{p-1} \max(((k_{i+1} - 1)G + g_{c(i+1)} + o_{scn(i+1)}), l_{nni}) + l_{nnp} + o_{rnp} + o_{rgbp} + o_{rdp}$$

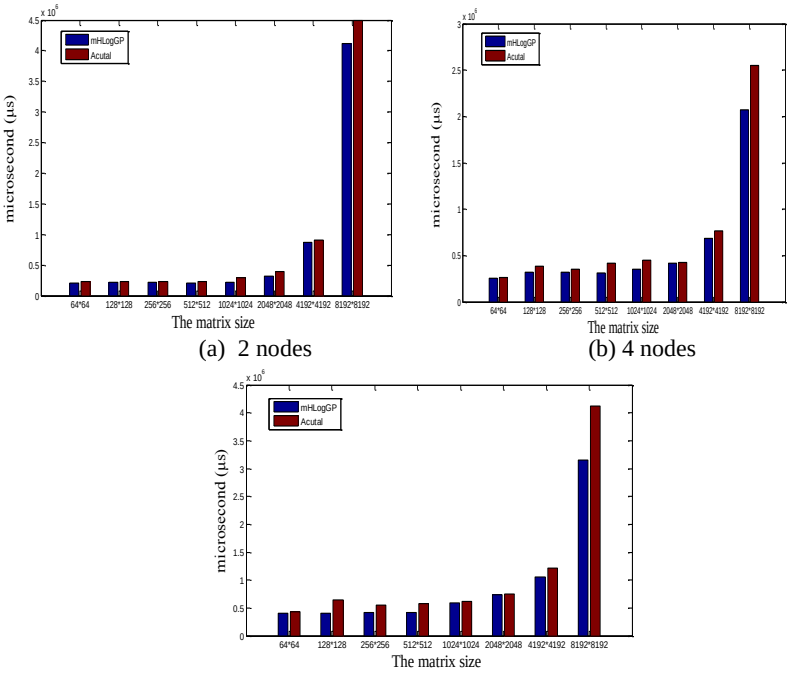
The cost of receiving messages (for the master process) is:

$$T_{ptop2} = \sum_{i=1}^{p} (o_{sgi} + o_{scni} + (k_i - 1)G + l_{nni} + o_{rni})$$

The cost of the broadcast Communication (for the master process) is:

$$T_{broP} = (\lg P + P - 1) * o + 2 * k * (P - 1) / P * (G + l_{nn})$$

The value of parameters is based on the experiments data in section 4. For single GPU, the computing cost and total cost is shown in Table 4. The computing time in matrix multiply is very short compared to communication time, so it is ignored.



(a) 2 nodes (b) 4 nodes

(c) 8 nodes

Fig. 5. The comparison of the predicted and actual execution time

The experiments are done on a cluster with 8 nodes including GPU. The comparison of the predicted and actual running time is showed in Fig.5. It is shown that our model could get competitive accuracy. With the increase of computation scale, the errors of the predicted running time increases. But, at worst, when matrix size becomes 8192*8192 on 8 nodes, the error of our model is 15-25%.

6 Conclusion And Future Work

Nowadays, CPU/GPU heterogeneous computing is a tendency. The conventional computation models do not fit in with the emerging CPU/GPU heterogeneous computing environment. In this paper, a new model named mHLogGP is presented. In mHLogGP, communication and memory access is abstracted by considering the characteristic of the CPU/GPU heterogeneous computing environment. This parallel computation model can be used to estimate the communication time and running time of

parallel program, to find the bottleneck and guide the optimization of parallel programs in heterogeneous computing environment. The evaluation experiments show that this computation model is valid. In the future, the computing time of parallel program will be concerned. We will also study the execution time of concurrent parallel programs under the heterogeneous computing environment with dynamic load.

Acknowledgment

This research was partially supported by the National High-Tech Research and Development Plan of China (Grant No. 2009AA01Z142).

References

1. Steven F., James W.: Parallelism in Random Access Machines. In: Proceedings of the 10th Annual Symposium on Theory of Computing, 1978:114-118 (1978)
2. Leslie G. V.: A bridging model for parallel computation. In: Commun. of the ACM, vol. 33, p. 103–111, 1990 (1990)
3. David C., Richard K., David P., Abhijit S., Klaus E. S., Eunice S.,Ramesh S., Thorsten v. E.: Logp: Towards a realistic model of parallel computation. In: Proc. of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (1993)
4. Albert A., Mihai F. I., Klaus E. S., and Chris S.: LogGP: incorporating long messages into the logp model áone step closer towards a realistic model for parallel computation. In: Proc. of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures (1995)
5. Thilo K., Henri E. B., and Kees V.: Fast measurement of logp parameters for message passing platforms. In: Proc. of the 4th Workshop on Runtime Systems for Parallel Programming (RTSPP) (2000)
6. Qasim A., Samuel P. M., and Vijay S. P.: Modeling advanced collective communication algorithms on cell-based systems. In: Proc. of the 15th ACM SIGPLAN symposium on Principles and practice of parallel computing (2010)
7. Kirk W. C., Rong G., Xian-HeSun: $\log_n P$ and $\log_3 P$: accurate analytical models of point-to-point communication. In: distributed systems. In: IEEE Trans. Computers, MARCH 2007,56(3):314-32 (2007)
8. Filip B., Xizhou F., Kirk W. C., and Dimitrios S. N.: Modeling multi-grain parallelism on heterogeneous multicore processors: A case study of the cell be. In: Proc. of the 2008 International Conference on High-Performance Embedded Architectures and Compilers, Goteborg, Sweden (2008)
9. Liang L., Xingjun Z., Jinghua F., and Xiaoshe D.: mPlogP: a Parallel Computation Model for Heterogeneous Multi-core Computer. In: Proc. of 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (2010)
10. Sunpyo H.，Hyesoon K.: An Analytical Model for a GPU Architecture with Memory-level and Thread-level Parallelism Awareness. In: ISCA'09, Proceedings of the 36th annual international symposium on Computer architecture, 2009: 152-163 (2009)
11. N. P. Karunadasa and D. N. Ranasinghe: Accelerating High Performance Applications with CUDA and MPI. In: Proc. Of Fourth International Conference on Industrial and Information Systems, ICIIS 2009, 28 - 31 December 2009, Sri Lanka (2009)