2015 The 5th International Workshop on Computer Science and Engineering
doi: 10.18178/wcse.2015.04.103

Multi-Core-OpenMP-SRs platform in Hard Real Time System

Furkan Rabee[1,2], Yong Liao[1], Maolin Yang[1],Jian Liu[1],Ge Zhu[1]

[1]School of Information and Software Engineering, University of Electronic Science & Technology of China (UESTC), China 2006 Xiyuan Ave. 611731, Cheng Du, China

[2]Computer Science Department, University of Kufa, Najaf , Iraq.

forkanr@yahoo.com, liaoyong@uestc.edu.cn, maolyang@gmail.com.cn

Keywords: Hard real-real time systems; multi-core-OpenMP-SRs platform; OpenMP.

Abstract. OpenMP has been designed as a programming model for taking advantage of multi-core architecture to do parallel programming, in this paper we present a new platform for multi-core hard real time system, by getting the advantage of OpenMP parallel programming to design OpenMP unit in hard real time system to be alternative for the GPU in free GPU systems; where this platform divided into three parts; first: clustering the multi-core-shared resources, second; constructing the OpenMP unit from the available cores, and third; designing the mechanism which govern the OpenMP unit with whole cores and shared resources. The performance of this platform compared it with classical multi-cores platform with MPCP locking protocol to support this protocol by the new design.

Introduction

Nowadays, Quad-core, multi-core & GPUs have already become the standard for both workstations and high performance computers. These systems use aggressive multi-threading so that whenever a thread is stalled, waiting for data, the thread can efficiently switch to execute another thread. Achieving good performance on these modern systems requires explicit structuring of the applications to exploit parallelism and data locality [11]. Programming Interface (API) such as OpenMP parallel programming, in the other hand multi-GPUs can be interfaced to on-chip multicores as efficient accelerators to increase system-level computational performance and improve system responsiveness [1, 2]. Multi-core technology offers very good performance and power efficiency and OpenMP has been designed as a programming model for taking advantage of multi-core architecture [11]. Depends on availability of cores and other workload parameters. This paper addresses these issues by supporting parallel programming in the hard real time system as substituted for the GPU, where it is done by: first; generating OpenMP parallel programming unit to simulate the GPU, and second; by suggesting a mechanism to this unit with similarity of GPU execution mechanism.

Related work

This paper developed two features: First; organizing the multi-cores-OpenMP-shared resources platform, and Second: executing mechanism to the tasks on that platform. Add to that, using the locked protocol to acquire shared resources.

The Multiprocessor Priority Ceiling Protocol (MPCP) [12] which favours priority queuing when tasks contend for locks, the Flexible Multiprocessor Locking Protocol (FMLP) [5] which favours FIFO queuing instead. Elliott and Anderson [7] also developed the Optimal *k* exclusion Global Locking Protocol (O-KGLP) under global scheduling. Glenn et al.[14] presented two GPU real-time analysis methods, addressing real world platform constraints, those two methods designed a mechanism of mutual exclusive to execute the tasks with GPU by treating it as shared resource. But those methods were used in soft real time system. In this paper would depend real-time mechanism could use in hard real time system. Glenn et al.[10] presented a framework for managing GPUs in multi-GPU multicore real-time systems, which provides flexible mechanisms for allocating GPUs to tasks; enables task state to be migrated among different GPUs, it is also enabled a single GPU's different engines to be accessed in parallel to the sporadic task model, even when GPU drivers are closed-source; and provides budget policing to the extent possible, given that GPU access is non-preemptive. This synchronization method depend on clustered CPUs/GPUs to CPU clusters and GPUs clusters; to reduce the complexity and migration time.

Task Model. A set of n sporadic tasks $T = \{T_1, T_2, \ldots, T_n\}$ are scheduled on multi-CPUs- shared resources platform that contain m CPUs, $p_{1,} p_{2,\ldots} p_m$. All task Tasks in T are indexed by decreasing order of priorities, so $i < j$ represents that the priority of T_i is higher than that of T_j.

Shared Resources(SRs). Tasks may share serially reusable resources (such as co-processors, I/O ports, or shared data structures, etc.). The task set contains of q local shared resources $\Phi_l = \{l_1, l_2, \ldots, l_q\}$ and w global shared resources $\Phi_g = \{g_1, g_2, \ldots, g_w\}$ Each task T_i is considered to be consists of two segments of time: first, non- critical section execution segment contain many jobs called *N_jobs*, calculated by $\sum_{j=1}^{a} N\_job_{i,j}$. Second, critical section execution segment called *C_jobs*[1], calculated by $\sum_{q=1}^{b} C\_job_{i,q}$, where:

a is number of non-critical execution segments of T_i.
b is number of critical section execution segments of T_i.

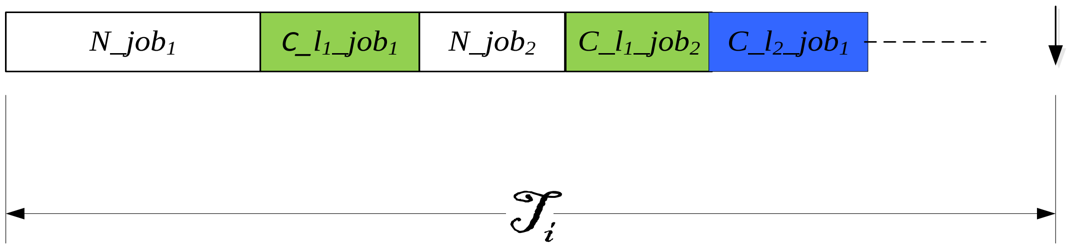


Figure 1: task jobs classifications

T_i : the period of T_i , D_i: the absolute deadline. In this paper supposed that $D_i = T_i$, *P(T):* set of tasks allocated on one processor.

The WCET of T_i is denoted by:

$$WC_i = \sum_{j=1}^{a} N\_job_{i,j} + \sum_{c=1}^{b} C\_job_{i,q} \quad (1)$$

Both the non-critical and critical sections called *jobs.*

OpenMP Unit Design. OpenMP (Open Multi-Processing) is an API that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, it does by implementation of multithreading, a method of parallelizing whereby a master thread (a series of instructions executed consecutively) forks a specified number of slave threads and a task is divided among them. The threads then run concurrently, with the runtime environment allocating threads to different processors [24]. According that we grouped some of multi-cores to OpenMP units (if it require) where each unit contain at least four processors.

OpenMP Unit Operations. We simulate the OpenMP unit to be like GPU operation, where There are two types of OpenMP operations. *Kernel* operations; are programs executed by the OpenMP unit, and other operations do by CPU host. Figure 2 shows the *OpenMP program execution sequence*, where a program running on a CPU initiates OpenMP unit. At time t_1, the CPU request to migrate to OpenMP, at t_2, the CPU get the acceptance to migrate. The OpenMP start executing its kernel at time 3. Finally, the kernel would finish the execution at t4 and request to the CPU to migrate back and get the acceptance at t5 and no longer required the OpenMP at time t_6. The time between t_2 and t_6 called *OpenMP critical section*. The OpenMP program execution sequence caused *mutual exclusive* mechanism, it is mean non-preemptive process.

[1] The critical section in this paper all *C_jobs* could acquire any shared resource except the GPU.

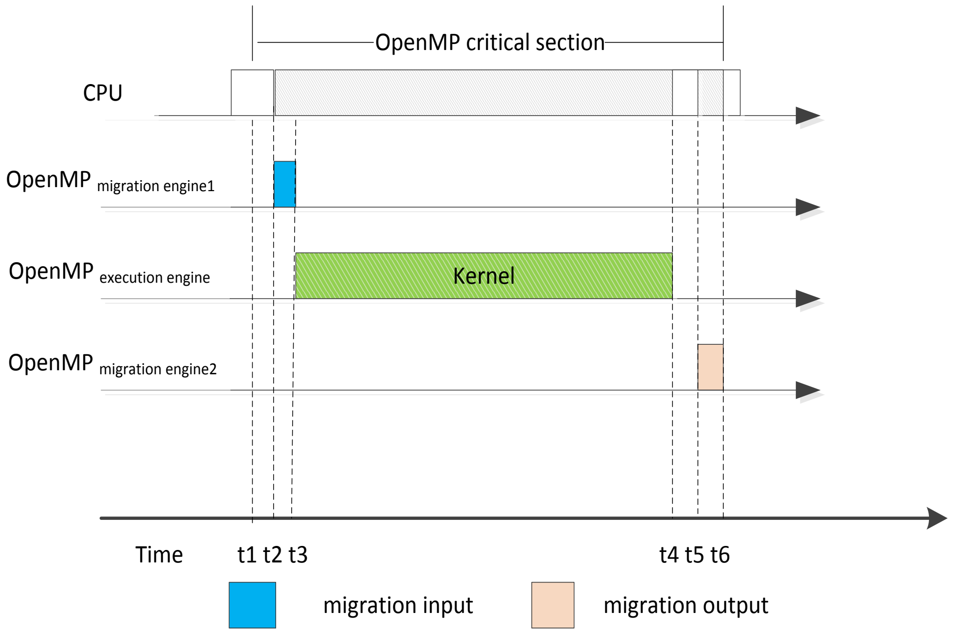


Fig. 2: OpenMP program execution mechanism sequence

Multi-Core-OpenMPs-SRs platform

The hybrid Multi-Core-OpenMPs-SRs platform is divided into two parts; the platform design, and the scheduling execution mechanism.

Multi-Core-OpenMPs-SRs platform design. This platform consists of the following sets:

1. Multi-core system.
2. Many of shared resources SRs.

First of all, the above sets should be distributed for at least two clusters, and then constitute the rest of platform. Figure 3 shows the platform design.

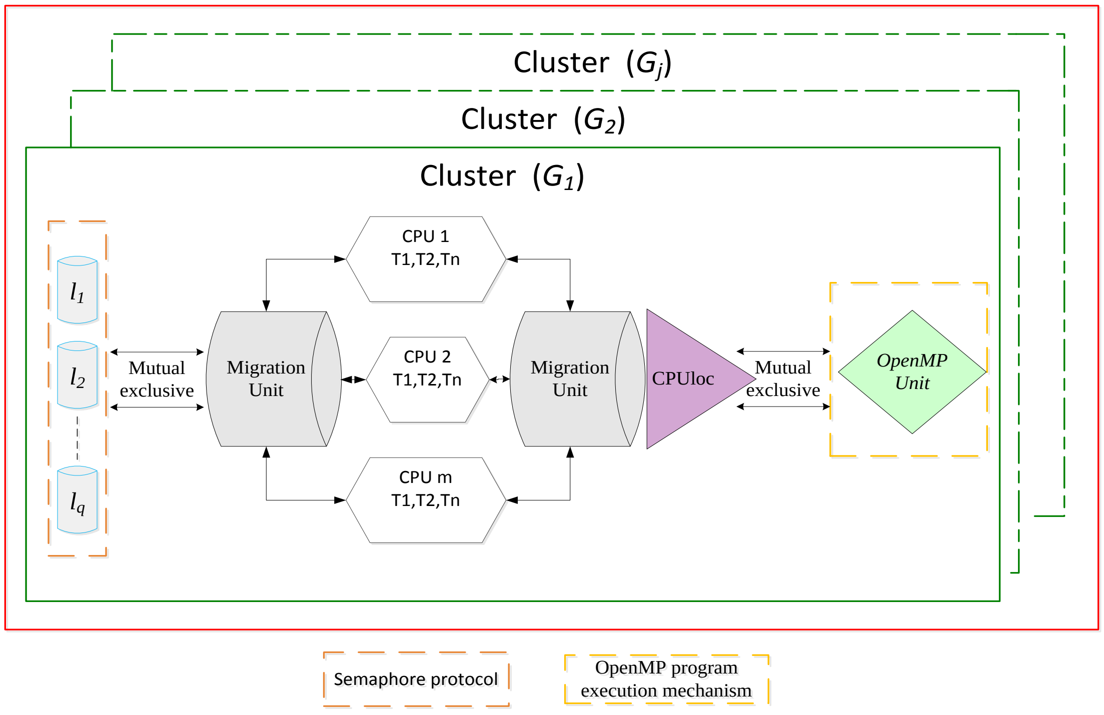


Fig. 3: Platform design.

Cluster Design. Each cluster has multi-cores and shared resources where grouped numbers of cores to construct the OpenMP unit.
where;

$$\underset{mo\geq 4}{op} = \sum_{x=1}^{mo} p_x \qquad (2)$$

$$\underset{mp\geq 4}{P} = \sum_{x=1}^{mp} p_x + \sum_{f=1}^{M} op_f \qquad (3)$$

where; the M refers to maximum numbers of OpenMP units in the system.

- Let G_j denote the *jth* hybrid cluster and *m=mo+mp*, where :

$$\underset{m\geq 8 \wedge k \geq 0}{G_j} \equiv P_m{}^j \wedge \sum_{z=1}^{q} l_z^j \qquad (4)$$

- This cluster has not GPU, but with OpenMPs unit would be alternative for GPU.
- The shared resources would be governed by MPCP protocol.

Cluster Organizations. The proposed cluster organized as follows:

C1. To construct OpenMP unit, it is should has at least four processors, and one cluster may has many OpenMP units..

C2. In case of CPUs < 8, each cluster at least contains one OpenMP unit.

C3. Each cluster has one CPU task locator (CPUloc): this allocator is one CPU from the CPUs in one cluster which is responsible for allocating the tasks to the OpenMP unit, or allocating the jobs to any other CPUs.

Multi-Core-OpenMPs-SRs scheduling and execution mechanism

This mechanism give a new scheduling and execution method for hard real time system where this method designated that; only the *N_jobs* could execute by OpenMP unit, and this execution we called it *acquired*. This mechanism has many rules and it is as follows:

R1. Any *N_job* released first, own CPU will be CPUloc. Or there are many *N_jobs* released in the same time, the CPU of higher priority job would be CPUloc, and this job would be acquire the OpenMP unit.

R2. Any one of *N_jobs* or *C_jobs* in one cluster whenever these jobs issue any request (to release, to execute, to acquire the shared resource, or to acquire OpenMP unit), must issue request to CPUloc first where this locator would be the responsible for allocating and migrate the *N_jobs* to the OpenMP unit.

R3. The CPUloc has another responsibility; where it has the decision which job could migrate to any CPUs/OpenMP units or still in own CPU, and this decision does in migration unit.

R4. Any job could execute on its CPU or migrate to another CPU in one cluster in case of its CPU is busy (suspend state or busy wait) and the other CPU is free.

R5. The CPUloc would be in *busy-wait* state when the GPU acquired by any *N_job*.

R6. The CPUloc would be in *suspend* state when the OpenMP acquired by any *N_job*. It is mean any *C_job* could executed within suspending time.

R7. The requests issued to OpenMP unit by CPUloc would be pass in two queues; initially by priority queue, and then in FIFO queue. Figure 4 shows the CPUloc structure and its queues.

R8. Acquiring the local shared resources locked by MPCP protocol.

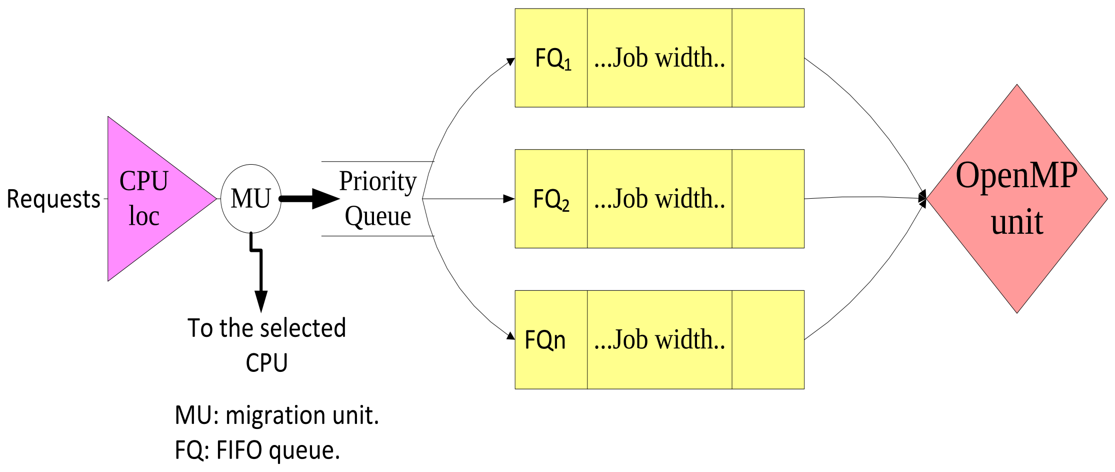


Fig. 4: CPU allocator structure design

Experimental Work

This section described the evaluations for the experimental work where it did by two stages: implementation stage and simulation stage where the simulation depends on OpenMP performances which calculated by the implementation. The simulations evaluate schedulability performance for the system.

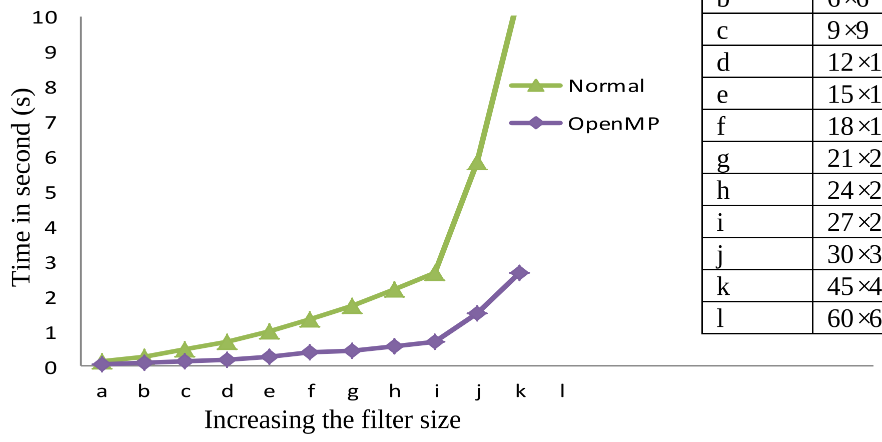
This paper worked on hard real-time schedulability which determined by EDF-schedulability[23] which calculated by the following equation:

$$U_i = \sum_{i=1}^{n-1} (WC_i + B_i) / per_i \leq D_i \tag{5}$$

where $per_i = T_i$. The non-critical section which acquire the OpenMP unit[2], divided by 9 depending on our evaluation in implementation part. We supposed that all *N_jobs* for all tasks would acquire OpenMP kernel for worse case and applied it in equation 5.

Implementation Setup. To get a precise evaluation, we did some implementations to check the performances for the CPU and OpenMP unit, where we did these implantations on Intel® core™ i5-2450M CPU@ 2.5 GHz and 4 G RAM. implemented on 64-bit Ubuntu operating system. And figure 5 show the CPU and OpenMP performances. This implemented did the convolution for (900×900) matrices with different filters size once by CPU sequential methods and the other by OpenMP. This CPU generate four thread to implement the OpenMP programing We found the OpenMP unit outperformed the CPU in normal case about 9.

[2]Where we supposed in all experiments the OpenMP unit contains four processors and four threads.



symbol	Filter size
a	3×3
b	6×6
c	9×9
d	12×12
e	15×15
f	18×18
g	21×21
h	24×24
i	27×27
j	30×30
k	45×45
l	60×60

Fig. 5: CPU and OpenMP performance

Experimental setup. Our experiments did by evaluating the ratio of schedulable task sets; where for each task set we tested it by three phases: four, six, and eight CPUs plus one OpenMP unit for each task set with each phase. We described factors that affect the schedulability as follows:

- The total system utilization U_i which represents the work load of the system vary from:
  1. 0.4 to 4 (step 0.2) for four CPUs.
  2. 0.4 to 6 (step 0.4) for six CPUs.
  3. 0.4 to 8 (step 0.4) for eight CPUs.
- Task periods are generated from [50ms, 1000ms] with uniform distribution.
- System utilization has been selected cap (0.4,4) in each scenario via four CPUs with one OpenMP once again. Tasks had been generated as follows:
  1. Light tasks from 0.01 to 0.4.
  2. Heavy tasks from 0.4 to 0.8.

Experiments consist of the above parameters and the results of all experiments show a total of 210 scenarios.

Priorities are assigned to tasks according to partition EDF [23]. A total number of 3050 task sets are generated for each scenario, and were tested using different schedulability algorithms.

Result and Discussion. In this section we compared the performance of our proposal with MPCP performance. The proposed method outperforms and other method in all of the scenarios shown in figure 6 to figure 8. Each processor contains more than one task, since in figure 6 (a and b); the platforms consist of 4CPU- one OpenMP unit-2SRs, the platform shows in figure 6 (c) consists of 6CPUs-one OpenMP unit -2SR, and the platform shown in figure 6 (d) consists of 8CPUs-one OpenMP unit -2SRs.

1. Figure 6 shows the ratio of schedulable task sets with two critical sections per task. The critical section length is 5% WC_i, the maximum numbers of shared resources for the set are two.
   - The task utilizations in figure (a) were light utilizations. With increasing the total system utilization caused reduce the schedule task sets. Since the performances our proposal outperformed clearly on the other and they closed to each other when the total system utilization about 2.8.
   - The task utilizations in figure (b) were heavy utilizations. With increasing the total system utilization caused reduce the schedule task sets. Since the performances our proposal outperformed clearly on the other and they closed to each other when the total system utilization about 1.2 first and then diverge and returned to close in 4.
   - The task utilizations in figure (c) were light utilizations for 6CPUs-one OpenMP unit. With increasing the total system utilization caused reduce the schedule task sets. Since the performances our proposal outperformed clearly on the other event they reached to 6.

- The task utilizations in figure (d) were light utilizations for 8CPUs-one OpenMP unit. With increasing the total system utilization caused reduce the schedule task sets. Since the performances our proposal outperformed clearly on the other and then closed to together in 6.4.

2. Figure 7 shows the performance of the methods when increasing the length of critical sections; this was applied on 4CPU- one OpenMP unit -2SRs with light tasks utilization. The total system utilization is 1.2. The scheduled task sets decreased with increasing the length of critical section because increasing the blocking time and this effect on the performance of response time for the tasks. The increasing of the critical section length start form 5% to 25% step 1.
3. Figure 8 show the performance of each method with increasing the number of shared resource. This was applied on 4CPU-one OpenMP unit and the total system utilization is 1.2, where each critical section length is 10% WC_i, and the maximum number of shared resources was ten. The scheduled task sets decreased with increasing the length of critical section because increasing the blocking time and this effect on the performance of response time for the tasks.

Conclusion and Future Works

In this paper, we present *Multi-Core-OpenMPs-SRs platform*, which could be implemented on system scheduled by partitioned scheduling, and this is first platform substitute the GPU by OpenMP unit in multi-core hard real time system. This platform can solve many of the researchers Fears from the closed source GPU and made it easy to use. Our experiments on that platform did for many phases of task set with only one OpenMP unit, it was outperform the traditional platform with well-known MPCP protocol, and if increase the number of OpenMP unit, we can get more efficient and utilized system. For future work, we will try to implement this policy by designing a kernel can simulate the scheduling of this platform by C language program to be online with advantages of this platforms and then trying to applying it on a real life application.

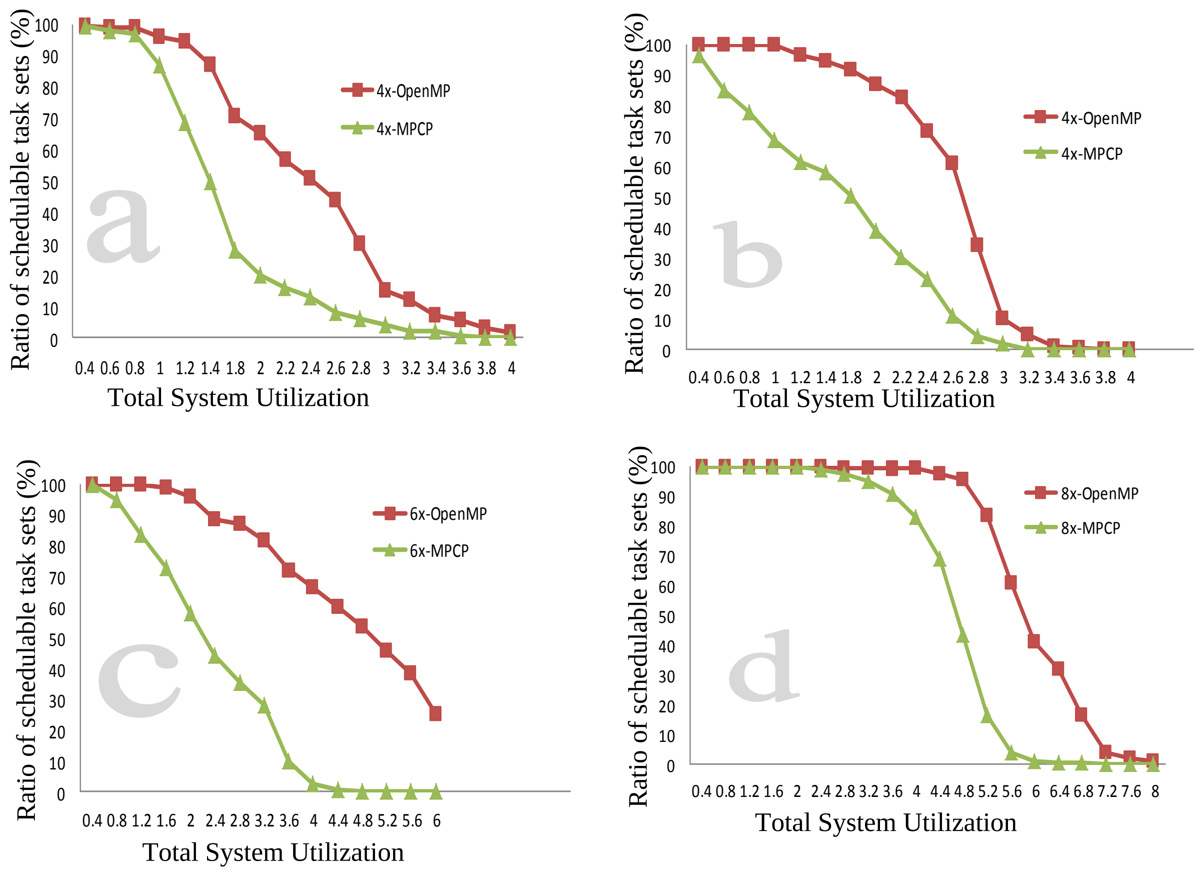


Fig 6: Performance with increasing total system utilization

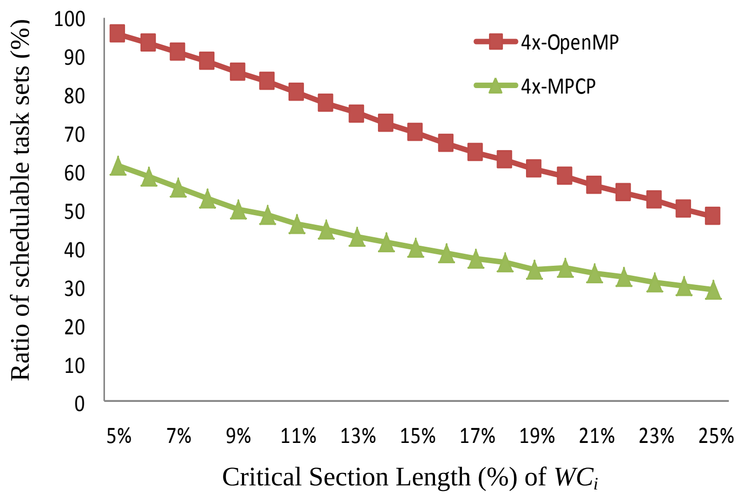


Fig. 7: Performance with Increasing critical section length

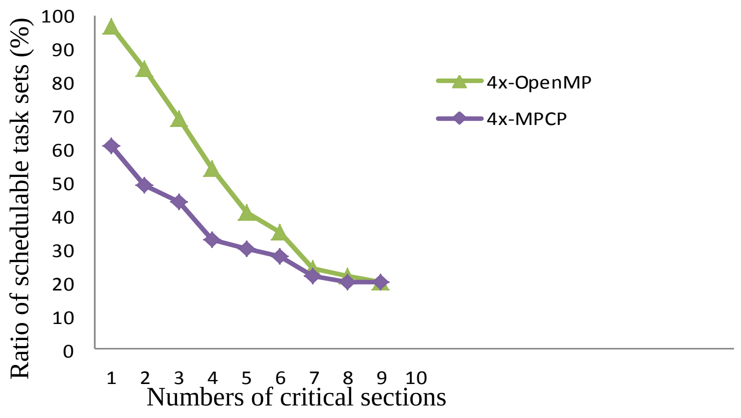


Fig. 8: Performance with increasing numbers of shared resource

Acknowledgements

This work has been supported in part by the National Natural Science Foundation of China (Grant No. 61103041), the Fundamental Research Funds for the Central Universities of China (Grant No. ZYGX2012J070), the PhD student Academic Support Program of UESTC (Grant No. YBXSZC20131028), the Huawei Technology Foundation (Grant No. ZYGX2012J070), and the National High-tech R&D Program of China (Grant No. SQ2011GX02D03708).

References

[1] A. Acosta, V. Blanco, F. Almeida, “Towards the dynamic load balancing on heterogeneous multi-GPU systems,” in IEEE Parallel and Distributed Processing with Applications (ISPA’12), pp. 646-653,(Jul. 2012).

[2] Z. Ziming, V. Rychkov, A. Lastovetsky, “Data partitioning on heterogeneous multicore and multi-GPU systems using functional performance models of data-parallel applications,” in IEEE Cluster Computing (Cluster’12), (Sep. 2012), pp. 191-199.

[3] G. Elliott, J. Anderson, “Globally scheduled real-time multiprocessor systems with GPUs,” Journal of Real-Time Systems, vol. 48, no. 1,(2012), pp. 34-74.

[4] G. Elliott, J. Anderson, "Robust real-time multiprocessor interrupt handling motivated by GPUs," in IEEE Euromicro Conference on Real-Time Systems (ECRTS' 12),(Jul. 2012), pp. 267-276.

[5] Block A, Leontyev H, Brandenburg BB, Anderson JH" A flexible real-time locking protocols for multiprocessors," the 13th IEEE Real-Time Computing Systems and Applications, (2007),pp 47-56.

[6] B. Ward, G. Elliott, J. Anderson, "Replica-request priority donation: a real-time progress mechanism for global locking protocols," in IEEE Embedded and Real-Time Computing Systems and Applications (RTCSA' 12), (Aug. 2012), pp. 280-289.

[7] G. Elliott, J. Anderson, "An optimal k-exclusion real-time locking protocol motivated by multi-GPU systems," Journal of Real-Time Systems, vol. 49, no. 2, pp. 140-170, (2013).

[8] H. Leontyev and J. Anderson. "A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. Real-Time Systems," 43(1):60–92, (September 2009).

[9] Mikael Asberg, Thomas Nolte, and Daniel Hallmans " Towards using the Graphics Processing Unit (GPU) for Embedded Systems," IEEE 17th Conference on Emerging Technologies & Factory Automation,(2012), pp 1-4.

[10] Glenn A. Elliott, Bryan C. Ward, James H. Anderson" GPUSync: A Framework for Real-Time GPU Management," in 34th IEEE Real-Time Systems Symposium (RTSS), (2013), pp 33-44.

[11] Krishnahari Thouti, S.R.Sathe "Comparison of OpenMP & OpenCL Parallel Processing Technologies" the 30th IEEE Real-Time Systems Symposium, (2009), pp 469-478.

[12] Rajkumar R, Sha L, Lehoczky JP "Real-time synchronization protocols for multiprocessors," the 1988 IEEE Real-Time Systems Symposium,(1988), pp 256-269.

[13] Liu CL, Layland JW "Scheduling algorithms for multiprogramming in a hard real-time environment," J ACM 20(1) ,(1973), pp 40-61.

[14] Glenn A. Elliott and James H. Anderson "Globally Scheduled Real-Time Multiprocessor Systems with GPUs,"Journal of Real-Time Systems, vol. 48, no. 1,pp. 34-74, (2012).

[15] Vincent Boyer, Didier El Baz, Moussa Elkihel "Dense Dynamic Programming on Multi GPU," in 19th IEEE International Euromicro Conference on Parallel, Distributed and Network-Based Processing,(2011), 545 – 551.

[16] Maolin Yang, Hang Lei, Yong Liao, Furkan Rabee. "PK-OMLP: An OMLP based k-Exclusion Real-Time Locking Protocol for Multi-GPU Sharing Under Partitioned Scheduling," in 11th IEEE conference on Embedded computing, (2013).

[17] Shinpei Kato, Karthik Lakshmanan, Aman Kumar, Mihir Kelkar, Yutaka Ishikawa, Ragunathan (Raj) Rajkumar. "RGEM: A Responsive GPGPU Execution Model for Runtime Engines," in 32nd IEEE Real-Time Systems Symposium,(2011), pp 57 – 66.

[18] Al ́ecio P. D. Binotto_ y, Bernardo M. V. Pedrasy, Marcelo G ̈otz_, Arjan Kuijpery,Carlos E. Pereira_, Andr ́e Storky, and Dieter W. Fellnery. "Effective Dynamic Scheduling on Heterogeneous Multi/Manycore Desktop Platforms," in 22nd International Symposium on Computer Architecture and High Performance Computing Workshops,(2010).

[19] Konstantinos I. Karantasis, Eleftherios D. Polychronopoulos. "Programming GPU Clusters with Shared Memory Abstraction in Software," 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing,(2011),pp 223 – 230.

[20] Rajkumar R "Real-time synchronization protocols for shared memory multiprocessors," the 10th Distributed Computing Systems,(1990), pp 116-123.

[21] S. K. Dhall and C. L. Liu, "On a Real-Time Scheduling Problem," *Operations Research,* vol. 26, number 1, pp. 127-140, (1978).

[22] Brandenburg BB, Anderson JH "an Implementation of PCP, SRP, D-PCP, M-PCP, and FMLP real-time Synchronization Protocols in LITMUS," the 14th IEEE embedded real-time computing system and application, (2008).

[23] C. L. Liu and J. W. Layland,"Scheduling algorithms for multiprogramming in a hard-real-time environment," Journal of ACM, 20(1),pp. 46–61, (1973).

[24] http://en.wikipedia.org/wiki/OpenMP.

[25] OpenCL Programming Guide for the CUDA Architecture. Available from: http://www.nvidia.com/content/cudazone/download/OpenCL/