Pakistan Journal of Science, 71 (4 Suppl.): 2019, Page 175-178
ISSN: 2411-0930

CRITICAL ANALYSIS OF 3D MANYCORE MEMORY ARCHITECTURE USING MAPREDUCE

A. Ali*, S. Talpur and R.B. Baloch

Mehran University of Engineering and Technology, Man Campus Jamshoro Pakistan.
*Corresponding author's E-mail: a.athar87@yahoo.com

ABSTRACT: Data-intensive applications have become one of the most important computer applications today due to increasing requirements of user demand. To get greater application performance it needs greater computing capability. The solution of this lies in MapReduce parallel programming model with many-core micro architecture. MapReduce was created by google for application development on data centers with thousands of servers. But the problem with present many-core micro architecture is that it does not match MapReduce runtime which makes many-core chip system unable to get deserving computing capability. In order to achieve huge computing and processing capability in high performance many-core chip system, this research combines many-core and cloud computing, two latest directions in computer science, in terms of computer micro architecture and studies how to design many-core 3D memory architecture to meet the demand of memory performance in micro architecture for MapReduce.

Keywords: MapReduce; Three-dimensional Memory Architecture; Network-on-Chip; Hash-associative Cache

INTRODUCTION

With the increasing popularity of multicore technology, the number of on-chip core has a rapid growth. Multicore platform presents a very broad field of application prospects. However, this is only possible through a parallel program to be able to make full use of and make consistent with the growth of the number of practical effects. Writing efficient parallel programs has never been easy. The programmers have to face problems including data distribution, scalability, load balancing, fault tolerance system and a large number of complex issues associated with parallelism. Authoritative research firm Gartner in 2008 lists the next 25 years, "seven major challenges facing the IT market" and put the era of multicore parallel programming in second place. Data-oriented parallel to the MapReduce programming model is undoubtedly the best answer to this challenge. Through reasonable abstraction to the application programmer to hide parallelism related issues.

Existing data-oriented parallel programming model has the support for mostly cluster based platform designed and implemented. The existing nuclear chip system cannot match the MapReduce runtime. Stanford University proposed the multicore MapReduce implementation Phoenix (Berezecki *et al.*, 2011). Phoenix is based on memory shared multicore platform to achieve the MapReduce runtime. The operation of the intermediate results and data became a key to Phoenix performance. There are no network connection problems between the computers and large MapReduce runtime performance bottlenecks on the cluster system. The performance bottleneck merging < key, value > data structure adopted by this embody in multicore/the nuclear chip system was due to for lack of cache and memory access. Embodiment for whole intermediate data < key, value > must be in the middle of the map and reduce phase to identify the map in the map phase aaccording to the order of the input data and reduce required the input data must be grouped by the key. We observe how quickly it completes for the map output to reduce the input of the processing of the intermediate result. These are organized as shown in Figure 1 in the middle of the buffer data structure. Literature (Tsoutsouras *et al.*, 2018) for existing problems using Hash and B + tree data structures to MapReduce intermediate < key, value > is optimized and in smaller handle auditing (16 processor cores) case achieved good effect, but with the doubling of the number of cores processing increases. The solution of these problems only from the aspect of software can't do it. At the same time, the many-core system inherits conflicts of sharing resource access and communication link congestion, numerical control information, mixed control signal and transmission delay communication. Solving the problems of the system performance must be from the micro level on the structure of the system (Zhang and Cao, 2017).

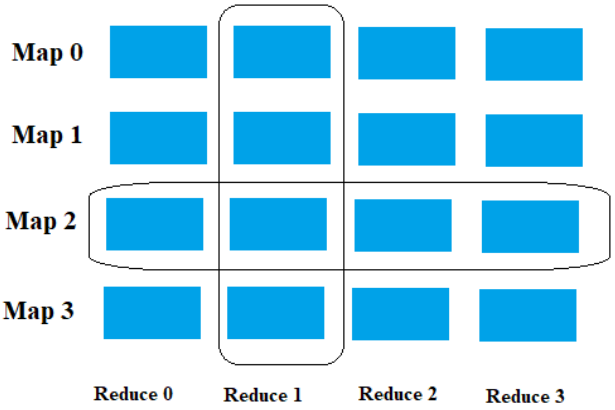

Fig. 1. Phoenix Intermediate buffer data structure

MATERIALS AND METHODS

Proposed by the many-cores system as shown in figure 2, by 1 of handling core layer and multiple cache layers. Each processing by L1, L1 ccontroller and the router encapsulated into tile structure. Cache layer and core layer contains the same number of nodes. Core layer uses the 2D mesh piece of interconnection structure and cache layer in each node uses half the company commander interconnect for each interconnection. It constitutes the so-called row and the column structure using 3D laminated between layers and vertical interconnection function block technology.

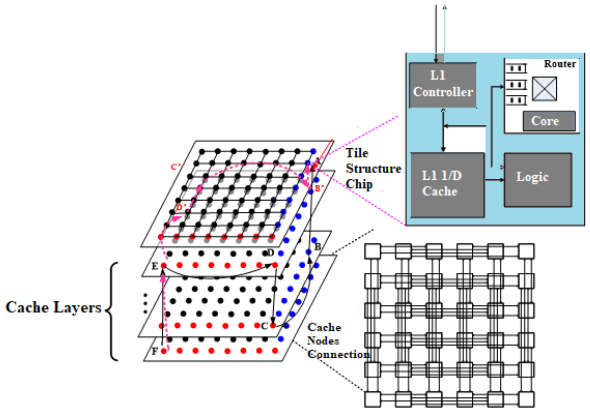

Fig. 2. General view of many-core 3D storage architecture

Long-term interconnect design can ensure that any piece of data in the cache back to the core's maximum path in diameter is 5 jump. Reducing the core access data latency ITRS data shown that in 2011 3D IC technology support layer stack chips together, all the many-core system could support up to 500 processing cores and interconnection between 10 Cache layer with a small radius of network to deal with small delay to access the cache. This is what makes us ponder for the realization of the large capacity cache. When dealing with core number 500, a total of 10 cache layer of cache blocks for MapReduce runtime provides a large enough to cache capacity.

Cache layer on-chip network design: In 3D mesh network in order to achieve the ideal of 3 jump in diameter, any two of each layer between two nodes must have a direct connection. Our solution is added as a layer between the vertical interconnection cost. The length of interconnect and the network diameter is reduced to 3 from 5 jumps. Due to vertical interconnection the layers have the characteristics of high speed, low power consumption. At very low power consumption and latency increase in exchange for several multiple reduced wiring scale. Note that our scheme by using a three-dimensional on the network of all lines within a vertical plane or all columns of the node between two nodes is realized by using long interconnect and interconnection between different nodes on the scattered to different layer on the surface.

As shown in figure 2, F, E, C, D nodes make lines in the form of surface forming full interconnection C, D, B, A, B ` etc. Due to vertical interconnection cost between the layers is small, a row or a column of all wired together into one layer between all nodes. Making sure that on the level of the stack to achieve the interconnection between the nodes. For example, in Figure 2 node F, as the layer of the bank's need 7 to interconnect, but we put it in our design 7 with interconnection line distribution in each layer. E nodes directly connected with the edge of the node. The benefit of distribution of the length of interconnection to different routing nodes, on the basis of what our scheme is feasible.

Mapreduce communication mode and channel design: MapReduce runtime is the transmission of mass data for the core processing. At the same time, the master thread needs each worker thread in real-time. Core has to fetch and control information that has the characteristics of short and real-time requirements, suitable for 2D mesh transmission. The cache data has the characteristics of large amount of data, suitable for long interconnection between cache layers and the core layers. For vertical channel of transmission, we put two channel respectively referred to as "control channel" and "data channel". A variety of transport channels provision can facilitate according to different types of transmission content to choose the appropriate network. Due to the fact that in many-core system level L2 cache is much larger than the L1 level within the cache, the cache data between different levels gets in and out frequently. MapReduce runtime is the execution time, the master and the map and reduce work between threads information through the control information transmission channel, such as the scheduler for scheduling information of all the work. The Work returns in the running state of information

transmission and data storage location information, etc. The processing requires to Map and Reduce and later merge process. The input and output of data are done through "data transmission channel".

MapReduce runtime is the execution of large quantities of data in the process of concurrent processing. large amount of data need to access system frequently which requires micro architecture provides a larger bandwidth and large capacity cache system. MapReduce runtime needs temporary intermediate < key, value > and in the middle of follow-up phase to < key, value > output after merge. So the middle < key, value > should exist in the cache until the merge work is completed as far as possible in order to save the cache space and improve the MapReduce runtime performance (Majzoub *et al.*, 2019; Li *et al.*, 2016).

By increasing the number of cache layer nodes in many-core system for MapReduce runtime provides a large enough capacity to cache. Within cache block, each layer of block has a LLC/Directory Controller, memory and disk, which are based on a distributed directory protocol to maintain data consistency. Single diagram is shown in figure 3.

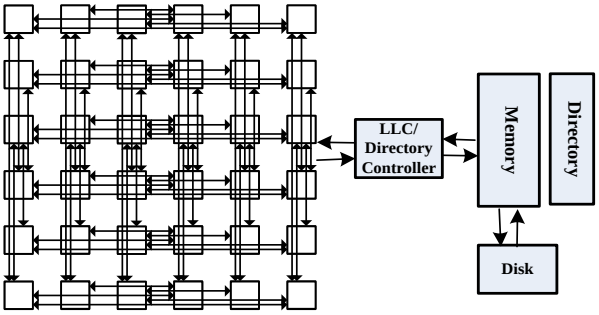

Fig. 3. single-layer Cache design

For accessing the data, the processing core first visits L1 cache. Cache controller will access requests to the cache layers. Each cache layer LLC / directory controller retrieve data after receipt. It then uses the data channel to a distance of not more than 5 hops transmitted to the disposal of requester to achieve the purpose of data access.

In cache management strategy, tile encapsulated with L1 cache and L2 level cache, there is no hierarchical relationship between them. Cascade memory controller is able to take full advantage of the benefits of L2 cache. Cache does not exist between the different levels with the same data stored in causing a waste. L2 cache data is sent to the requester when dealing with core directly through the 5-hop 3D data channel for transmission. There is no need to write along the cache which is different from the "data migration" management approach (Khalil *et al.*, 2013; Ge *et al.,* 2019).

Thus in this way it reduces power consumption and latency overhead. During runtime execution MapReduce frequently executes large amount of data and give output as the write (write) operation. The output (write) operations are mainly Map and Reduce intermediate output which are the merged output. As there are n-way set-associative indexes for each index cache group of n lines will all be activated and requires relatively tag n times. Due to the frequent output (write) operations resulting in large amount of data generated by frequent comparative mapping activities will surely consume more power consumption, which is not suitable for the use of existing groups associated technologies i.e. MapReduce. Proposed hash associated cache technology will compare only one time after each write operation. This is bound to greatly reduce power consumption.

RESULTS AND DISCUSSION

We put all the layers of cache data blocks with unified coding. Since for MapReduce runtime, each time data for a new map task is to be processed according to the load allocation part (or all) cache thread to the map execution before the cache allocated for data cleanup. We Select the appropriate Hash function, the data to be processed as a factor for the resulting <key, value> stored sequentially through the Hash function to the cache. Once the Map phase is completed, reduce phase plays its part, it accepts the task threads, respectively for the different layers of Cache, same space incorporates the data in the Cache, a strategy is adopted as the reference layer Cache data. The data are written to the other layers to be merged. Of course, in between the layers it must be considered that implementing network load situation, you can use different Reduce select for different threads and Cache layer as the base layer to balance the network load. Finally, a unified final merge output is generated.

By evaluating the performance of system with parameters given in Table I, increasing the number of Cache layers results in greater Cache capacity which helps in matching MapReduce runtime execution and gives greater computing capability as in Figure 4.

Table I. Test system parameters.

Cache Line	Cache Bank Block Size	Cache Total Capacity	Replacement Policy
64 bytes	512/bank 10 cycles	2500 M	LRU

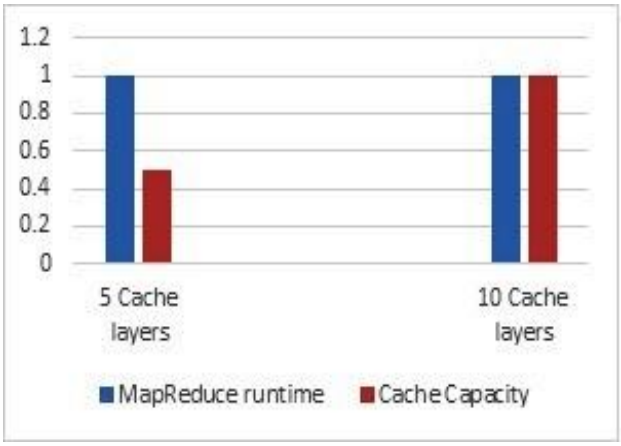

Fig. 4. Cache Capacity to match MapReduce runtime

In addition, the system has advantages in the following aspects:

- Dual-channel transmission communication mode based on the contents of different communication channels has improved data sharing between nuclear transfer speeds. Cache layer interconnection cable with plug length decreases interconnection network diameter.
- It reduces chip count nodes communication through routers, thereby reducing the average on-chip communication delay and power consumption, provides high-capacity cache design (Yan *et al.*, 2015).
- There are multiple layers of Cache Directory Controller channel constitute a distributed Memory. To solve memory bus congestion, in each layer of a Cache Directory Controller any core is able to access cache and it is directly connected to each core. Different memory access path cores constitute a high-bandwidth and many choose the Cache help to resolve the conflict.
- Hash associated Cache reduces set-associative Cache multiple comparison problem which causes overhead power consumption. This improves the operational performance and saves Cache memory space (Xing *et al.*, 2018).

Conclusion: This paper evaluated the use of multiple layered cache memory in order to match the MapReduce runtime. We introduced multiple layered Cache in many-core micro architecture which provided enough cache capacity for MapReduce which take input and generate output concurrently and frequently. Moreover, using hash-associated cache technology has helped in reducing power consumption because it compares the frequently generated outputs of MapReduce only once.

Overall this work establishes that by introducing 3D memory Cache in many-core memory architecture it could get huge computing capability by matching MapReduce runtime in cloud computing environment.

REFERENCES

Berezecki, M., E. Frachtenberg, M. Paleczny and K. Steele (2011). Many-core key-value store. In *2011 International Green Computing Conference and Workshops* (pp. 1-8). IEEE.

Ge, F., L. Wang, H. Lu, N. Wu, F. Zhou and Y. Zhang (2019). STT-RAM Based Energy-Efficient Hybrid Cache Architecture for 3D Chip Multiprocessors. *Engineering Letters*, *27*(1).

Khalil, S., S.A. Salem, S. Nassar and E.M. Saad (2013). Mapreduce performance in heterogeneous environments: a review. *International Journal of Scientific & Engineering Research*, *4*(4), 410-416.

Li, L., E. Wang, X. Dong and Z. Zhu (2016). The Optimization of Memory Access Congestion for MapReduce Applications on Manycore Systems. *The Computer Journal*, *59*(3), 325-337.

Majzoub, S., R. A. Saleh, I. Ashraf, M. Taouil and S. Hamdioui (2019). Energy Optimization for Large-Scale 3D Manycores in the Dark-Silicon Era. *IEEE Access*, *7*, 33115-33129.

Tsoutsouras, V., S. Xydis and D. Soudris (2018). Application-Arrival Rate Aware Distributed Run-Time Resource Management for Many-Core Computing Platforms. *IEEE Transactions on Multi-Scale Computing Systems*, *4*(3), 285-298.

Xing, Y., F. Liu, N. Xiao, Z. Chen and Y. Lu (2018). Capability for Multi-Core and Many-Core Memory Systems: A Case-Study With Xeon Processors. *IEEE Access*, *7*, 47655-47662.

Yan, Q., F.R. Yu, Q. Gong and J. Li (2015). Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges. *IEEE Communications Surveys & Tutorials*, *18*(1), 602-622.

Zhang, Y. and H. Cao (2017). DMR: A deterministic MapReduce for multicore systems. *International Journal of Parallel Programming*, *45*(1), 128-141.