Operating system in the era of the many core chips

Dániel Kristóf Kiss*†
*Doctoral School of Applied Informatics, Óbuda University, Budapest, Hungary Email: kiss.kristof@phd.uni-obuda.hu
†Mentor Graphics Corp., Budapest, Hungary, Email: daniel_kiss@mentor.com

***Abstract*—Era of the many core ships will begin soon. Multi core chips are already mean a challenge for the whole software industry. In this paper the problems are details from the view point of the operating system. A new operating system architecture is introduced to efficiently support many core chip based systems.**

I. Hardware architectures

The multicore era is started on all platform. The era was begun in 2006 on the PC platform. In 2007 Intel researchers reported [8] the creation of a single chip with 80 computational units reaching 1.28 Tera-FLOPs. This means the beginning of the many core era. One of the most recent Intel research is the so-called Single Chip Cloud Computer. It contains 24 processing units and connections between them. This architecture brings a new kind of constraints. It is important to note that some of the processors are dedicated to special activities, e.g. graphics or bus interface.

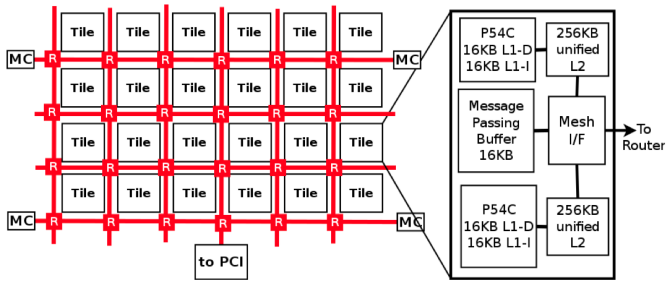

Fig. 1. SCCC architecture [7]

Figure 1 shows the topology of the Single Chip Cloud Computer. Tile contains two CPUs with dedicated L2 cache and an message buffer. R stands for the message router. MC means Memory Controller. This architecture is message based only. Communication cost is depends on the location of the participants and the load of the communication bus. Please note that these parameters are not controlled usually by initiator of the communication. The same behavior could be observed in a non-message based processor. We assume that all kind of architecture could behave as a message base system. Some similar many core architecture are under development [2]. These architectures make new challenges in the software industry.

II. Level of parallelisation

In this section the parallelism levels and aspects will be discussed from the operating system view point in multi and many core systems. Theoretically there are four level of parallelism. The first is the bit-level parallelism. Word size of the processor defines that level. If the processor word size is increased then the required cycles for an operation on of the word size data is decreased. The second level of the parallelism is the instruction level. This level is introduced by the superscalar processors. The third level is the data level parallelism. That means that the same operation performed parallel on different data. The fourth level is the task level parallelism. The operating system is provides the that level. In this section that level will be detailed.

Most of the parallel computation research focus on the single problem solving in a parallel way. While GPU based parallel computing is efficiently supports that find problem solving then the many core systems are different. Consider the following real word situations. Any desktop operating system with a typical user usually runs a few hundred processes. Most of these processes just serve the user applications. The other example could be any web server which servers concurrently thousands of users. In the future maybe one core on the server serves only one user because the number of the cores is more than equal with the number of the users.

Other aspect of the parallelism is the fact some of the algorithms are sequential, or the non parallelizable parts are too huge.

Operating system point of view the parallelism is the threads of an given process. Note that the number of the threads is not enough information to describe the need of the process. Threads are created and destroyed dynamically at runtime while the threads may block each other. The factor of the maximum utilization of the process is could be defined by the number of how many core could be used concurrently.

III. Operating system architectures

In an multi core system the operating system can easily manage resources. Some of the operating system already maintain a process queue for each processor to decrease the inter processor communication during the task switching. Start this section with the exact and precise definition for the operating system. Prof. Tanenbaum in his book wrote that it depends on who speaks about the operating system. There are two viewpoints the top-to-button and button-to-top. The first view point defines in that way : the operating system extends the real hardware. The second view point defines that operating system is a resource handler. In our opinion both statement are correct and valid. The current popular operating systems like Windows, Linux, etc. are delivered in a package,

978-1-4673-5206-2/12/$31.00 ©2012 IEEE

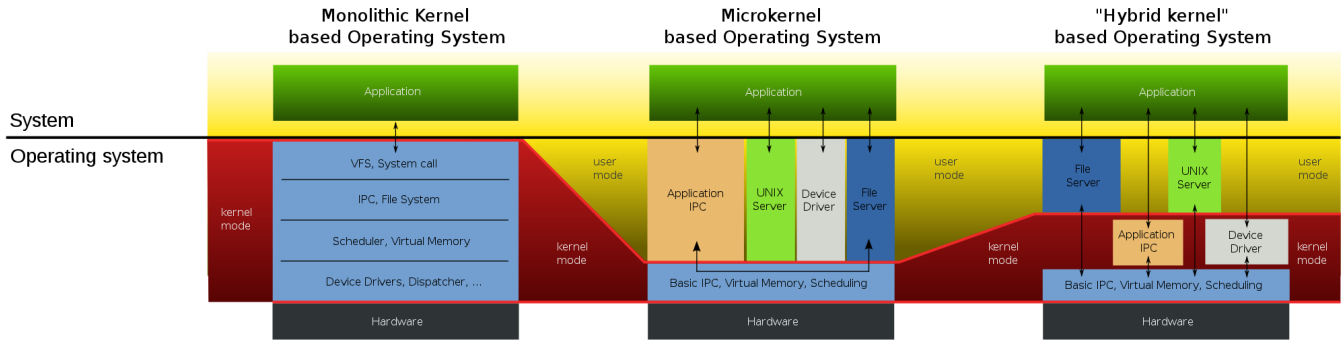

Fig. 2. Operating system architectures [3]

which contains lots of programs. Most of these programs are not part of the operating system. Different system has different terminology because of the different architecture, applied technology and some historical reasons. There are three classic operating system architectures, figure 2 show the major differences.

The operating systems are categorized into three groups. These categories are the Monolithic, Micro and Hybrid. There are good examples for each categories. The architecture of the operating system is important because it has effect on the whole system. More details could be found in a previous article [4].

Besides of the system security and reliability the other important aspect is the schedulibility of the system. This parameter comes more and more important because currently the OS components have to run on different processors. Monolithic operating system is just a big program couldnt run it exactly parallel using multiple processors. Basically micro and hybrid kernels now have step advantage because most of the services are separated into different tasks but the real challenge (the main aim of our research) is to find the best methods for the optimal execution. These systems intensively use the kernel services to send receive messages, lock objects. For this research the micro kernel based operating system architecture is selected.

IV. Operating system with multiple kernel

Operating system has own data to describe the state of processes, memory usages. These data structures must be kept consistent. Parallel computing is about how to share the data between the concurrent processes. In case of an many core system the cost of a locking and transfer between nodes is expensive. Many core architectures use message passing between the processor cores while others share the global memory. The common in these methods is the bandwidth usage. For the easier understanding the message passing architecture is assumed in this paper. That is also assumed in this paper the Operating System is microkernel based and only message passing used for inter process communication.

Processes are considered connected if there is any communication between them. The strength of the connection is an important factor of the distribution problem. Let define a cost function that represents the strength of the connection. This function depends on the message repetition rate and the size of the messages.

$$S_{P_a P_b} = \sum_{i=0}^{i<messages} \lfloor F_{strength}(M^i_{size}, M^i_{rate}) \rfloor \qquad (1)$$

Let $S_{P_a P_b}$ stand for the connection strength between the processes P_a and P_b. Furthermore, let M represent a message with two properties, namely the size and the transfer rate. $F_{strength}$ is an increasing continuous function. The characteristic of this function should reflect to the properties of the underlying hardware. This metric could be used to determinate the optimal distribution of the processes in a many core environment [5].

For a given system these parameters are known usually or easy to collect that information. Connection to a memory controller or a peripheral interface could be defined in similar way. Let message node stand for any message sender/receiver component in the system.

For the process distribution a graph is used where the weight of an edges is the connection strength and the nodes are the process and message nodes. Lets define three class of message nodes. Processes belong to the first class, these nodes are relocatable between processors. System services belong to the second class. The third class is the non relocatable hardware components. For example the memory interfaces, network interfaces or other peripherals. The biggest difference between the class of processes and the services is that the services are accessed with symbolic access while processes are accessed directly always. Symbolic access means when a services is accessed then the accessor does not want to communicate with an exact instance of the service it only want to access to a service which able to serve the its request. For example in a given system the network load is high then it maybe would be useful if the network stack could be multiplied to serve more request parallel. Two instance of the networking stack may double the capacity. In the class of the processes two types are differentiated. First is the process internal nodes like threads and inner communication. The other is the process and its outer communication.

Kernel is a component in the operating system. It is responsible for the context switching, scheduling, memory management and the inter process communication. The number of the processes and the used memory by the processes requires a lot of memory in the kernel side.

The goal is to increase parallelism inside the operating system which will decrease the inner dependencies and improve the system performance. Only the kernel is not classified. The kernel could act as a service from that point of view multiple instance is could exist in a same system. For example on a system with dual processors each processor has own kernel. Please note that the system still has one operating system. The kernel is responsible for only its own processor and the associated process. Any request to the other core is a cooperation between the kernels. An message based system there is two type of kernel to kernel interaction: message passing to an process of the other kernel and administration.

The figure 3 shows an graph of processes. Red node represents the memory controller. Other nodes in the graph represent a process. The edges represent connection between the processes while the thickness of the edge represents the connection strength of the connection. K1 and K2 are kernels and the oval shapes mark a possible subset of the processes.

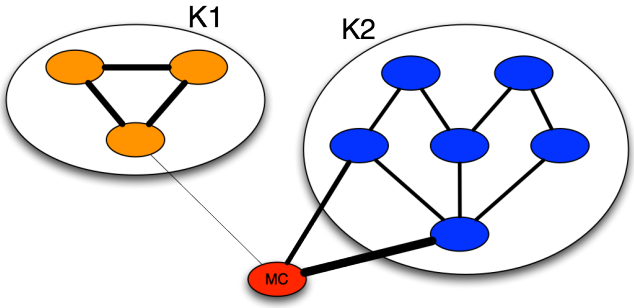

Fig. 3. Graph of processes

The figure 4 shows an possible distribution and core allocation of the graph on the figure 3.

The graph of the processes is continuously changing during the execution. The goal is to find a suitable sets of processes where the interactions mostly happens inside the set. This set could supervised by an dedicated kernel. To find that set the graph should be divided into subgraphs. The division could made on the edges where the weight is below then a given limit. This is essential to minimize the kernel to kernel interaction. Analysis of these graphs show some interesting behavior. Sometimes the groups are formed around services or devices. For example the memory interface magnetize the processes, the network interface magnetize the network stack. While devices fix components in the architecture the services are not. Sometime the subgraphs remain huge around a service then as mentioned below the system could initiate a fork for the service.

Duplication of the service is a modification of the graph. Modification of the graph may effect the optimal placement

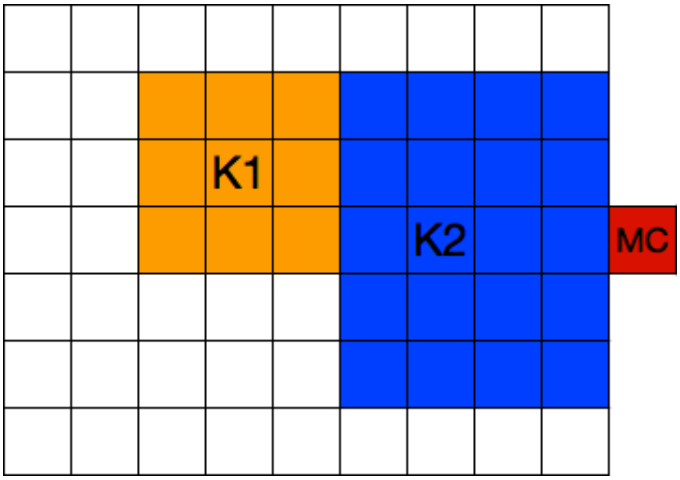

Fig. 4. Possible distribution of the processes

[5].

The multi kernel architecture may effect on the connection strength calculation, decrease the cost of the local traffic but increase the global traffic cost. The transition from a formation to another has calculable cost which must be higher then the benefit of the transformation. Prediction and control the transitions the behavior based prediction could be used [6]

V. Notes on the Implementation

In this section a walk through is given on the implementation aspect of the introduced method. In case of microkernels three major modules should be discussed. These modules are tightly coupled. First part is the process and thread management, the second part is the memory management and the third is the inter process communication. The target dependent part of these modules are not discussed. In this article the inter process communication is detailed due to the introduced method cause significant effect on it.

An unique number is used for identify the sender or the receiver of the IPC communication in case a single kernel system. This number is used for lookup the receiver. That will not work in a multi kernel system because it doesn't contain the enough information for the message rooting. The new message identifier(id) must meet with the following requirements. It shall able to describe any destination node in a unique way. To support multiplicable services the id shall able to describe the meta or symbolic destination nodes. The description shall independent from the actual location of the sender and the destination. This requires for the dynamic placement of the processes. The goal of lookup algorithm is that resolve the id to a processor identifier which is usually an row and column number pair. The id shall not changed during if the process is relocated or the supervisor kernel is splitted or merged. The database of the lookup may changed but the destination shall not be changed. Processes store the id during the execution that prevents the id modification during the execution.

Number of solution had been considered and here the best is discussed. The above mentioned requirements and constraints make the identification complex. To reduce that complexity and fulfill the requirements the proposal contains three level for the message id handling. Keep the existing APIs and provide simplicity to the applications the top level or process level ID remain an unique integer id. The id is unique in the scope of the process. The lowest layer is responsible for the message delivery to the target processor. This layer is hardware dependent. The middle layer transforms the id from the processor to the lower layer id. Identifiers in the middle layer are system wide unique identifiers. Each message contains that system identifier. Due to when a processor receives a message this identifier will be used to resolve the high level id for the process. The system identifier contains a unique process identifier and a logical location information. Logical location information describes the process location in the tree of the processes and kernel domains. The format of the system wide identifier is that: $< processIdentifier > . < path >$ where the path could consists of multiple segments.

This format seems to similar to the Domain Name System (DNS) but the paths are handled dynamically and modified. To highlight the differences the most important use cases are detailed here.

Let P_x is a processes and K_x is kernels while x is real number.

Initially let there are two processes in a single kernel. Table V shows this case.

Process ID	System ID	Core ID
1	$P1$.K0	0-0
2	$P2$.K0	0-0

TABLE I

The system goes to the state which is showed by the figure 3. $K1$ and $K2$ inherited from the K0 parent. Table shows the lookup table. This table is stored in both of the kernels.

Process ID	System ID	Core ID
1	$P1.K1$.K0	z-w
2	$P2.K2$.K0	x-y

TABLE II

As mentioned the lookup information is sent with the message. The purpose of this overhead to efficiently handle the case when the receiver of the message is not found at the destination. $K1$ kernel tries to deliver a message from $P1$ to $P2$. In the x-y position the local lookup table does not contain the information about the $P2$ process. Instead of sending back an error message the message is sent to the parent which is the $K1$ kernel. It may know the location of the $P2$ process for example it is moved to an other core or it is terminated. If $K1$ is not exists anymore then message is sent to the K0. Finally K0 will send back the error notification if needed. Update of the lookup table of the $K1$ kernel will be initiated by the reply of the message.

If a kernel is disposed then it is merged into an other kernel. The merge modifies the routing table. If the target kernel is the parent then the old tag from the path is just removed. Other case the tag of the path is replaced with the new kernel relative path. The type of the Id usually match with the word type of the processor. The initial kernel owns the full range of the Ids. A kernel manage its Id range in that way; the first half of the range is assigned to processes. The second half is reserved. It used when new range is requested due to the child kernel runs out from its own Ids or the kernel crates a new child kernel. On a 32 bit machine the id range is 0 to 4294967295. A system with 1000 kernels still has 2147483 ids per kernel. This number of ids are look more then enough. If an Id is reassigned to a different process the message to the previous process will not cause any problem because the new process may not receive message from the sender. Even it may receive a message from that sender then it shall open the port first which will reset the routing at the sender side. If a service has multiple instance then the rooting table points to the selected instance in each table.

VI. Further work

The introduced method created for many core computers. There is a strong conjecture; the method could be applied on every system where the computation units are connected. The messaging system may extended to support remote process communication. Subject of the further research to confirm that the proposed approach works in a reconfigurable environment.

Acknowledgment

The authors gratefully acknowledge the grant provided by the project TMOP-4.2.2/B-10/1-2010-0020, Support of the scientific training, workshops, and establish talent management system at the Óbuda University.

References

[1] Operating systems: design and implementation, Andrew S. Tanenbaum, Albert S. Woodhull, Pearson Prentice Hall, 2006 ISBN:9780131429383

[2] http://www.adapteva.com/products/epiphany-ip/

[3] http://en.wikibooks.org/wiki/Operating-System-Design/Kernel-Architecture

[4] Kiss, D.K.; Rövid, A.; , "Multi-core processor needs from scheduling point of view," Intelligent Systems and Informatics (SISY), 2010 8th International Symposium on , vol., no., pp.219-224, 10-11 Sept. 2010 doi: 10.1109/SISY.2010.5647499

[5] Kiss, D.K; Distribution problem in multi-processor environments,11th International Conference on Global Research and Education in Engineers for Better Life, pp.431-434, ISBN: 978-615-5018-37-4

[6] Kiss, D.K.; Rovid, A.; , "Theory for proactive operating systems,"Intelligent Engineering Systems (INES), 2012 IEEE 16th International Conference on , vol., no., pp.463-467, 13-15 June 2012 doi: 10.1109/INES.2012.6249879

[7] Ehsan Totoni, Babak Behzad, Swapnil Ghike, Josep Torrellas; "Comparing the Power and Performance of IntelsSCC to State-of-the-Art CPUs and GPUs" http://iacoma.cs.uiuc.edu/iacoma-papers/ispass12.pdf

[8] Vangal, S.; Howard, J.; Ruhl, G.; Dighe, S.; Wilson, H.; Tschanz, J.; Finan, D.; Iyer, P.; Singh, A.; Jacob, T.; Jain, S.; Venkataraman, S.; Hoskote, Y.; Borkar, N.; , "An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS," Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International , vol., no., pp.98-589, 11-15 Feb. 2007