ISSN No. 0976-5697

Volume 5, No. 2, March 2014 (Special Issue)

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

# PMP: Parallelism with Multicore Processors

Thatikonda Venkatesh[1], YVR Naga Pawan[2]
II M.Tech (SE)[1], Associate Professor, Dept. of IT[2],
Anurag Engineering College, Kodad, India

***Abstract:*** Multicore processors unleashed the computing power of desktops, laptops figuring out thermal, power, memory constraints. This paper throws light on multiprocessor architecture, implementing parallelism, task scheduling, and shared memory management

***Keywords:*** PMP, multicore, processor, Parallelism

## I. INTRODUCTION

As embedded applications become more and more complicated, embedded system designers rely more on multi-processor or multi-core platforms to obtain high computing performance. Meanwhile, due to the power/thermal constraints, the memory bottleneck, as well as the limitation of the instructional level parallelism in programs, industry is changing its gear toward the multi-core architecture rather than continuing to pursue high performance uniprocessor architecture. Conceivably, most of the future embedded systems will be built upon multicore architectures. A major issue in developing multi-core computing systems is how to utilize the available computing resources most effectively.

Recently, a new multi-core scheduling approach, i.e. so called semi-partitioned approach has been proposed. In the semi-partitioned scheduling approach, most tasks are assigned to one particular processor, i.e. the same as the partitioned scheduling approach[5]. However, a few of tasks (i.e. no more than $(M-1)$ tasks, where $M$ is the number of processors) are allowed to be divided into several subtasks and those sub tasks are assigned to different processors.

The semi-partitioned approach not only outperforms the traditional partitioned approach and global approach theoretically but also has been shown as sound and practical in the real implementation. Furthermore, by implementing the semi-partitioned scheduling method in the Linux operating system, and running experiments on an Intel Core-i7 4-cores computer, Zhang et al showed that the overhead in the task migration can be relatively low, and thus its impact on the schedulability is small.

Fortunately, there is an impressive gain in processor performance, due to advances in hardware technologies and also to innovation in processor architecture (i.e., how the processor is designed and organized to perform its computational tasks). One distinguish development is the introduction of parallelism in the architecture of the processors in the form of pipelining, multitasking, and multithreading leading to significant performance enhancement. As a result of that a new type of relatively low-cost and powerful multi-core processor is emerged and widely-used in present computers. A multi-core processor is a processor which contains two or more microprocessors (cores) each with its own memory cache on a single chip. The cores can operate in parallel and run programs much faster than a traditional single-core chip with a comparable processing power. Intel is now providing seven-core processors, which is known as i7-core processors; the number of cores will continue to increase as technology advances. And then next another major component of a computer is the software, which is categorized into operating system (OS) and application software. Traditional computer software is vastly improved to efficiently utilize the processing power of the emerged multi-core processors.

With the advent of multi-core processors, the importance of parallel computing is significant in the modern computing era since the performance gain of software will mainly depend on the maximum utilization across the cores existing in a system. It is necessary that tools exists to make the full use of the capabilities offered by the parallel computing. Though current parallelizing compilers support parallelization at loop level it is a hard task to detect parallel patterns and do conversions at the compiler level.

Using multiple processor cores on a single chip allows designers to meet performance goals without using the maximum operating frequency. Overall performance is achieved with cores having simplified pipeline architectures is relatively equivalent to single core solution..

There are four distinct paths to develop application software for parallel computers:

a. Extend an existing compiler to translate sequential programs into parallel Programs.
b. 2. Extend an existing language with new operations that allow users to express Parallelism.
c. Add a new parallel language layer on top of an existing sequential language.
d. Define a totally new parallel language and compiler system.

### *A. Extend a Compiler*

In this approach a parallelized compiler is developed to detect parallelism in the existing programs written in sequential language.

### *B. Extend a Sequential Programming Language:*

Unlike the above process here existing language is extended that allow users to write code in parallel mechanism.



CONFERENCE PAPER
Two day National Conference on Advanced Trends and Challenges in Computer Science and Applications
Organized by: Shree Vishnu Engineering College for Women, Bhimavaram A.P.
Schedule: 18-19 March 2014

### *C. Add a Parallel Programming Layer:*

Parallel program is having two layers. The first or lower layer contains the core of the computation, in which a process manipulates the data to produce the result. An existing sequential programming language would be suitable for expressing this portion of the activity. The upper layer controls the creation and synchronization of processes and the partitioning of the data among the processes. These tasks could be programmed using a parallel language (perhaps a visual programming language). And compiler translate this two-layer parallel program into code, this code is suitable for execution on a parallel computer.

### *D. Create a Parallel Language:*

The fourth approach is to give the programmer the ability to express parallel operations explicitly. One way to support explicit parallel programming is to develop 3 parallel languages from scratch. Occam is the language famous example for this approach. it supports parallel as well as sequential execution of processes and automatic process communication and synchronization[7].

## II. MULTICORE ARCHITECTURE

We studied three multi-core platforms, with significantly different architecture and memory organizations. The first is a 4-core Intel Core i7 processor. Each core supports two Simultaneous Multi-threaded (SMT) thread contexts. The cores have private 32 KB L1 and 256 KB L2 caches but share an 8 MB L3 cache. The second platform we use is a 48-core AMD Many-Cores machine. There are four CPU chips on the memory bus, each holding 12 cores. The chips are connected using AMD proprietary Hyper-transport 3.0 links. On each chip, the cores are located on two separate dies, with each die holding 6 cores. Each core has a private 64 KB L1 and 512 KB L2 caches, and shares 6 MB L3 cache with other cores on the same die. A specialized interconnect is used to connect the caches across dies. The cores have non-uniform memory access (NUMA) to different regions in memory and experience non-uniform latencies on cache hits to the L3 cache depending on whether the cache line is in the L3 cache of the same die or a remote die. The third platform is the Tilera TilePro64, an many-core architecture with 64 identical tiled cores. Tilera features low latency and high bandwidth communication fabric interconnecting the cores[4].

While there are a number of challenges to the design of multicore architectures, arguably the most challenging aspect of the transition to multicore architectures is enabling mainstream application developers to make effective use of the multiple processors. To address this challenge we consider in this section the techniques of thread-level speculation (TLS) that can be used to automatically parallelize sequential applications and transactional memory (TM) which can simplify the task of writing parallel programs. As is evident from the chapters that describe these techniques, their efficient implementation requires the close interaction between systems software and hardware support[8].

### *A. Single-Core Memory Architecture:*

A typical computer has a Random Access Memory (RAM), often referred to as the working memory, and also a smaller cache memory with faster access time in between the RAM and CPU, as shown in Fig. 1. Every data element that is requested from the CPU will be brought into the cache memory, if not already there. The idea of introducing an intermediate memory between the RAM and CPU is based on the observation of data temporal and spatial locality.

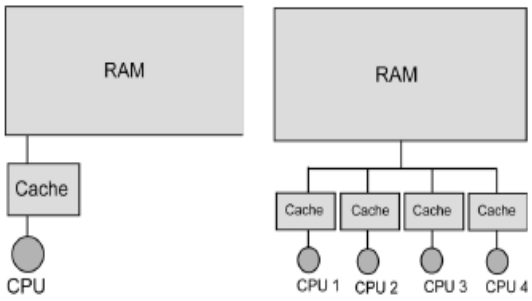


Figure 1: Single and Shared Memory Architecture

### *B. Shared Memory Multicore Architecture:*

In a shared-memory multicore architecture, there are several CPUs that can access a shared RAM, but each CPU also has a private cache memory, as shown in Fig. 1. Notice that here the CPUs share a common bus to communicate to the RAM. This is however only an example and the organization of bus connections differ among different computer architectures. The simplified view depicted in Fig. 1 shows that even though the CPUs have a shared memory, it is beneficial to construct an algorithm that will keep the data locally in the private cache memory of each CPU, improving the overall performance. Also it illustrates the fact that two processors can block memory accesses for each other since only one processor at the time can access the bus.

### *C. Parallel Processing Models:*

The first step in mapping an application to a multicore processor is to identify the task parallelism and select a processing model. There are two dominant models those are a Master/Slave model in which one core controls the work assignments on all cores, and the next one is Data Flow model in which work flows through processing stages as in a pipeline.

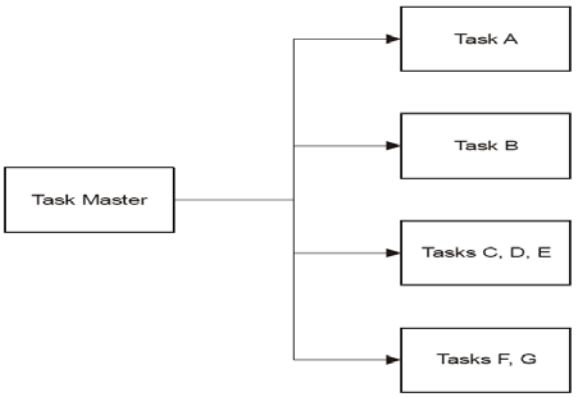


Figure 2: Master Slave Model

### *D. Data Flow Model:*

In the Data Flow model there will be a distributed control and execution. Here data is processed by each core using various algorithms and then the data is passed to



CONFERENCE PAPER
Two day National Conference on Advanced Trends and Challenges in Computer Science and Applications
Organized by: Shree Vishnu Engineering College for Women, Bhimavaram A.P.
Schedule: 18-19 March 2014

another core for further processing. The initial core is connected to an input interface and supplies the initial data for processing from either a sensor or FPGA. Applications those running on Data Flow model often contain large and computationally complex components that are dependent on each other and may not fit on a single core[9].

Synchronization of execution is achieved using message passing between cores. Data is passed between cores using shared memory or DMA transfers.

Data Flow processing is shown in Figure 3.

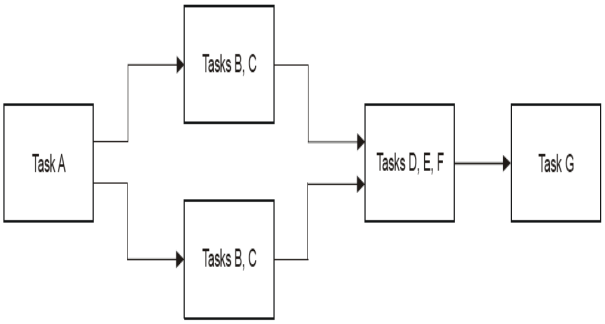


Figure 3: Dataflow Process Model

### *E. OpenMP Model:*

OpenMP is an Application Programming Interface (API) for developing multi-threaded applications in C/C++ or Fortran for shared-memory parallel (SMP) architectures. OpenMP standardizes the last 20 years of SMP practice and it is a programmer-friendly approach with many advantages. Using the API is very easy and quick to implement; once the programmer identifies parallel regions and inserts the relevant OpenMP constructs, the compiler and runtime system figures out the rest of the details. The API makes it easy to scale across cores and allows moving from an 'm' core implementation to an 'n' core implementation with minimal modifications to source code. And the OpenMP is sequential-coder friendly; that is, when a programmer has a sequential piece of code and he would like to parallelize it, it is not necessary to create a total separate multicore version of the program. Instead of this all-or-nothing approach, OpenMP provides an incremental approach to parallelization, where programmers can focus on parallelizing small blocks of code at a time. The API also allows programmers to maintain a single unified code base for both sequential and parallel versions of code[2].

## III. PARALLELISM WITH MULTICORE

### *A. Parallel Programming Models:*

Just as there are several different classes of parallel hardware, so too are there several distinct models of parallel programming. Each of them has a number of concrete realizations. OpenMP uses a shared-memory (or shared address space) programming model. In this model, as its name implies, that programs will be executed on one or more processors that share some or all of the available memory. Shared-memory programs are executed by multiple independent threads (execution states that are able to process an instruction stream); the threads share data but may also have some additional, private data. A different programming model has been proposed for distributed-memory systems[6].

Generically referred to as "message passing," this model assumes that programs will be executed by one or more processes, each of which has its own private address space. Message-passing approaches to parallel programming must provide a means to initiate and manage the participating processes, along with operations for sending and receiving messages, and possibly for performing special operations across data distributed among the different processes. The pure message passing model assumes that processes cooperate to exchange messages whenever one of them needs data produced by another one. However, some recent models are based on "single-sided communication." These assume that a process may directly interacted with memory across a network to read and write data anywhere on a machine.

Various realizations of both shared-memory and distributed-memory programming models have been defined and deployed. An ideal API for parallel programming is enough to permit the specification of many parallel algorithms, is easy to use, and leads to efficient programs. Moreover, the more transparent its implementation is, the easier it is likely to be for the programmer to understand how to obtain good performance. Some are a collection of library routines with which the programmer may specify some or all of the details of parallel execution (e.g., GA and Pthreads for shared-memory programming and MPI for MPPs), while others such as OpenMP and HPF take the form of additional instructions to the compiler, which is assumed to utilize them to generate the parallel code.

### *B. Automatic parallelization:*

Many compilers provide a flag, or option, for automatic parallelization of a program. When this is selected, the compiler analyzes the program and search for independent sets of instructions, in particular for loops whose iterations are independent of one another. And It then uses this information to generate explicitly parallel code. OpenMP directives enable the programmer to view and possibly improve the resulting code. The difficulty with relying on the compiler to detect and exploit parallelism in an application is that it may lack the necessary information to do a good job. For programs which are having simple structure, it may be worth trying this option.

### *C. MPI:[ Message Passing Interface]:*

The Message Passing Interface [MPI] was developed to facilitate portable programming for distributed-memory architectures (MPPs), where multiple processes execute independently and communicate data as needed by exchanging messages[3]. The API was designed to enable the creation of efficient parallel code, as well as to be broadly implementable. As a result of its success, it is the most widely used API for parallel programming in the high-end technical computing community, where MPPs and clusters are common. Since most vendors of shared-memory systems also provide MPI implementations that leverage the shared address space[1,10].

## IV. CONCLUSION

This paper envisages that the multicore programming models increasingly bring difference in desktop computing through multicore processors. The threading at software as well as hardware level boost the performance of computing.

The scheduling of the process in the multicore environment gives its able impact to computing performance and efficient memory management with various API's like OpenMP, MPI etc.

## V. REFERENCES


[1] Programming Massively Parallel Processors. A Hands-on Approach David B.Kirk and Wen-mei W. Hwu

[2] Parallel Programming in C with MPI and OpenMP: Michael J.Quinn Oregon State University

[3] Hybrid Programming with MPI: B.Estrade

[4] W-C. Feng and P. Balaji, "Tools and Environments for Multicore and Many-core Architecture," Computer, vol. 42, no. 12, pp. 26-27, Dec.2009.

[5] J.Dongara, I.Foster, G.Fox, W.Gropp,K.Kennedy, L. Torczon and A.White, The Sourcebook of Parallel Computing.Morgan Kaufmann Publishers, 2003.

[6] H. Kasim, V. March, R. Zhang, and S. See, "Survey on Parallel Programming Model, "Proc.IFIP Int'l Conf.Network and ParallelComputing , Vol 5245, pp. 266-275, Oct.2008

[7] M.J. Sottile, T.G Mattson, and C.E.. Rasmussen,Introduction to Concurrency in Programming Languages. CRC Press, 2010.

[8] G.R. Andrews, Foundations of Multithreaded, Parallel, and Distributed Programming Addison wesley, 1999.

[9] V. Kumar, A. Grama, A. Gupta, and G. Karypis, Introduction to Parallel Computing Design and Analysis Algorithms. Benjamin/Cummings Publishing Company 1994.

[10] W. Gropp, E. Lusk, and A. Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface, second ed.MIT Press, 1999.




CONFERENCE PAPER
Two day National Conference on Advanced Trends and Challenges in Computer Science and Applications
Organized by: Shree Vishnu Engineering College for Women, Bhimavaram A.P.
Schedule: 18-19 March 2014