A Survey on thread assignment techniques on Multicore Processors

Sandeep Lanka
Department of Electrical and Computer Engineering
Houghton, MI, USA
slanka@mtu.edu

***Abstract*—Threads can be assigned to multicore processors based on the Processor's Architecture to enhance performance. This paper surveys different efficient thread assignment techniques on processors with both Heterogeneous and Homogeneous cores. The first approach to assign threads is to assign threads statically depending on a prior knowledge. The other approach is to dynamically assign threads during execution. This paper classifies the thread assignment techniques based on whether they are assigned statically or dynamically on both Heterogeneous and Homogenous Architectures and their performance improvements from prior works.**

Keywords—Thread Assignment; Multicore processors;

I. INTRODUCTION

The main objective of a multicore architecture is either to improve the throughput and single thread performance or for power efficiency for the same workload. One approach is to increase simultaneous processing of multiple threads on the same processor by some means, that is, to increase Thread Level Parallelism. The other approach is to build multiprocessors that have same or different copies of more than one core that can handle threads simultaneously. If there is a lot of thread level parallelism that can be exploited in the given application, the second approach seems right. On the Other hand, if the application does not have that much Thread level parallelism it might not be a good idea to go for a Chip Level Multiprocessors Chip level Multiprocessors that have multiple copies of the same core are called Homogenous and multiprocessors that have cores that have different capabilities and performance levels are called Heterogeneous Multiprocessors. The need for Heterogeneous multiprocessor is the fact that different applications have different amounts of Instruction Level Parallelism(ILP). If the application has a large amount of Instruction Level Parallelism, a core that can issue many instructions per cycle such a Superscalar can perform really well but for a application with a very low amount of Instruction Level Parallelism(ILP) the same superscalar may not be as efficient. If there is another core that suits this kind of application, it is better that it is assigned to such a core. If the applications requirements are consistent we may want to opt for a homogenous multiprocessor. Prior works have shown that these kinds of architectures increase the performance significantly. The main problem with these kinds of multiprocessors is in assigning the threads efficiently to each core or even scheduling the threads or core switching based on the need. The assignment mechanisms must consider workload and behavior of threads overtime and decide the appropriate core in heterogeneous processors in case of dynamic assignment and map the threads to the corresponding cores. The other approach is to decide the priorities of the threads based on prior information and assign threads accordingly. The appropriate mapping of the threads to heterogeneous processors can increase the core utilization and thus the overall throughput which is the main aim of a multiprocessor.

This paper thus, deals with these thread assignment techniques to multicore processors for performance enhancement or power efficiency.

II. TERMINOLOGY AND TAXONOMY

Processors with Multiple cores integrated on the chips are called multiple processors. These are adressed as Chip level Multiprocessors(CMP). These cores can handle threads simultaneously and if the threads are distributed rightly, there is a scope for increase in throughput. The main problem is to rightly distribute threads and this paper deals with these techniques that increases the efficiency of these processors.

A. *Homongeneous/Heterogenous Multicore Processors*

The above mentioned Chip level multiprocessors can either be Homogenous or Heterogenous Multicore processors. Chips that use multiple copies of the same core are called Homogenous multicore processors. That means that all the cores are uniform with a fixed performance characteristics. Heterogenous processors on the other hand, have cores that are diverse. Core Diversity offers much higher abilty to adapt to the demands of the application.

1. *Static Assignment Techniques*:

Static Assignment techniques simply map the threads to the cores depending on the prior knowledge of the application or by some heuristic that does not depend on the dynamics of the application. Although these techniques might try to map threads based on the architecure, the thread-to-core mapping once made does not change between the execution of the program.

2. *Dynamic Assignment Techniques*:

   Dynamic Assignment techniques not only map the threads to the cores at first but also monitors the dynamics of the processor. Dynamic assinment techniques then map the threads to the cores depending upon the heuristic that determines which core is more capable for that part or to achieve a degree of parallelism or any other parameter.

3. *Fixed Assignment Techniques*:

   Application based Static or Dynamic Assignment techniques map the threads to the

4. *Application based Techniques*:

   Application based Static or Dynamic Assignment techniques map the threads to the cores depending on the prior knowledge of the application based

5. *Energy Efficient Techniques*:

   Application based Static or Dynamic Assignment techniques map the threads to the cores if the processors in a Energy Efficient manner.

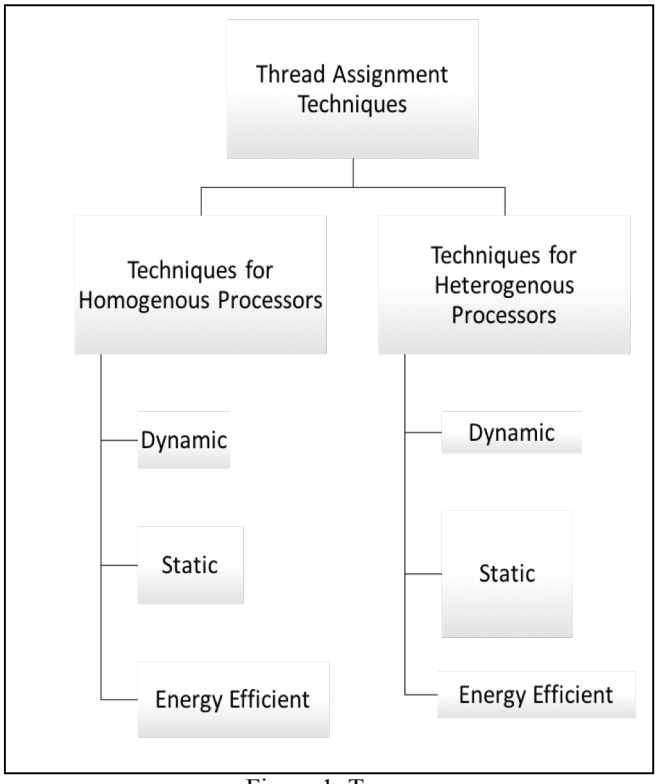

Figure 1: Taxonomy

III. THREAD ASSIGNMENT ON HOMOGENOUS PROCESSORS

If the application demands uniform performance throughout its execution and if there is a reasonable thread level parallelism associated with it, Homogenous processors can increase the throughput.

A. Static Techniques

I. Fixed

Fixed Thread assignment techniques are those that are the simplest assigning techniques that are consistent and the performance with respect to that assignment does not affect the scheduling mechanism. These techniques are general thread assignment techniques that just distribute threads to the resources.

1. *First In First Out:*

It is also known as First Come First Served technique. It is a simple technique which queues processes in the order which they arrive and assigns them to the cores one after another in a regular fashion. This might be very useful for aplications that constantly take a fixed time and thus mapping in this fashion increases the throughput. If the proceses take different times to execute and if there is a long process involved it increases the latency and thus the overall throughput.

2. *Fixed Priority Preemptive Scheduling*:

Each Process is assigned a rank and depending upon the rank, and the scheduler arranges the processes in the order of the their prority before scheduling them. Higher priority processes are scheduled first. If the number of rankings are limited, queues are ranked depending upon their priority and scheduler selects processes from low priority queues only after the higher priority queues are scheduled.

3. *Multilevel Queueing*:

   In this type of scheduling, processes are divided before hand, into categories that are considered different levels. For example, Common division is made between foreground and background processes and are scheduled seperately.

2. Application based

Application based assignment techniques are those that are based on the prior knowledge of the application and the assignments are assigned according to the heuristic that is application specific.

1.Global and Partitioned Multiprocessor Fixed Priority Scheduling with Deferred Preemption:
This method is proposed by Robert I Davis and Alan B , Jose M, Vincent N and Stefan M. P proposes a scheduling technique for a application on a homogenous multiprocessor system with some identical processors. The application is assumed to be a set of tasks and each task is assigned a unique priority. Tasks are assumed to comply with a *sporadic* task model. Each task comprises of a presumably unbounded sequence of jobs. Each job may arrive at any time once minimum time interval has elapsed since the arrival of the previous job of the same task.
Each task is characterized on the basis of its relative Deadline D, worst case execution time, C, minimum inter arrival time or period T. Each Tasks utilization is given by C/T.

This methods Priority Scheduling technique that is based on two prior different models of fixed priority scheduling. *Fixed Model,* also called cooperative scheduling

and and *Global Fixed Priority Scheduling*. In the fixed model introduced by Burns the non preemptive regions are recognized prior to execution. The *floating model* [Baruah 2005; Yao et al. 2009; Marinho et al. 2012] sets an upper bound to the length of the longest non preemptive region of any task.

Under Global Fixed Priority Scheduling with Deferred Preemption(gFPDS), it is focused on Global fixed priority Scheduling of an application on a homogenous multiprocessor system with m identical processors. It is assumed that all taska have constrained deadline wchich is D≤T; Each Task is assumed to have a FNR of length F in the range of [1,C]. Finding the appropriate FNR is assumed to be part of the scheduling problem. The worst case response time is the lonest time it take from the release of a task to its complete execution.

Under gFPDS, at any given time, the *m* ready tasks with the highest priorities are selected for execution. Final non preemptive regions are assumed to be implemented by manipulating task priorities. Thus, the task executing its FNR has the highest priority and will not be preempted.

The tasks are assumed to be independent and so cannot be blocked from executing by another task. Other than the due to contention for the processors. Each job may execute on at most one processor at any given time which means job parallelism is not permitted.

A task is set to be schedulable with respect to some scheduling algorithm, if all valid sequences can be scheduled without any missed deadlines.

A priority Assignment policy is said to be optimal with respect to schedulability test for some type of fixed scheduling algorithm if there are no task sets that are deemed schedulable, according to the test, under the scheduling algorithm using other priority ordering policy.

The paper that proposes the method further shows the comparison between gFPDS and other contemporary or prior algorithms and concludes that, gFPDS shows that appropriate choice of length of these preemptive regions can enhance performance.

B. **Dynamic Techniques**

1. Round Robin

Round Robin assignment is distributing threads to processor cores one after another in a round fashion. The tasks are blindly allocated to all the processor without any runtime considerations.

2. Thread Delaying and Thread Balancing:

Thes techniques are proposed by R Rakvic and Q.Cai, J. Gonzalez,G.Magklis, P.Chaparro and A.Gonzalez utlizes a mechanism called *meeting point thread characterization* that identifies the critical thread of a single multithreaded application as well as non critical threads. This is achieved by a thread counter that accumulates number of iterations executed for the parallel loop. At specific intervals each threads broadcast this information. This information determines the noncritical and the critical threads. Each threads own iteration counter and the other threads iteration counters difference and the counter of the slowest one. The goal is to scale down the voltage/frequency for the next interval time. This scaling down of Voltage/frequency keeping in mind the expected time of arrival is called Thread Delaying.

Thread Balancing is a hardware scheme for a simultaneous multithreaded processors running parallel threads. The goal is to improve execution time by speeding the critical thread. This is achieved by giving the most priority to the critical thread in the issue slots.

The important part to either perform Thread Delaying or balancing is to recognize the critical threads dynamically. This is achieved by checking the workload balance at intermediate points of a parallel loop. These intermediate checkpoints are called *meeting points*. The whole process consists of the three steps:

1. Insertion of the meeting points: This is chosen to be the place where all the threads visit many times during the parallel execution.

2. Identification of critical threads: A thread-private counter is incremented every time a core decodes an instruction encoding a meeting point. The user inserts the meeting point using a pragma which when decode increments the hardware counter of the thread.

3. Utilizing the criticality information: This information is used to implement Thread Delaying or Thread Balancing based on the requirement for energy efficieny or improve overall execution time.

Due to the cubic relationship between power and frequency/voltage, it is better to make non critical threads run at lower frequencies.

In case of Thread balancing, A simple fair issue might not be very useful if all threads are not similar and hence critical threads may need more priority than non critical threads. This method dynamicaly speeds up the slowest thread by increasing its priority. In order to implement this, we should be able to identify threads that are the slowest. This is the same as the thread delaying methodsto identify critical threads. A meeting point in the program is recognized and the meeting point is used by imbalancehardware logic that detects, at runtime, the imbalance.

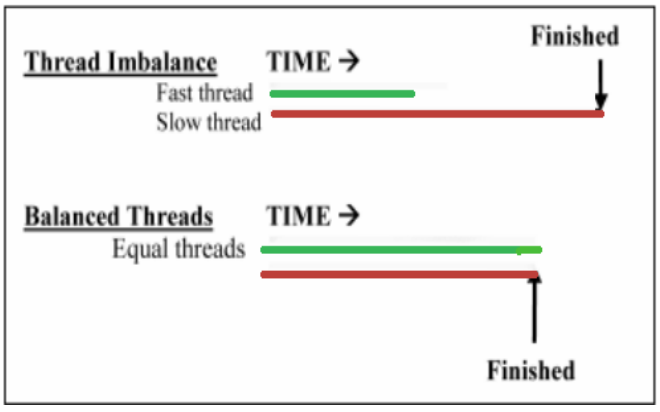

Figure 2: Thread Delaying

The imbalance hardware logic monitors the instruction that is currently executing to determine if the thread has reached its meeting point. It gives out two things , the slowest of the threads and the number of iterations the faster thread is faster than the slower of the two threads.

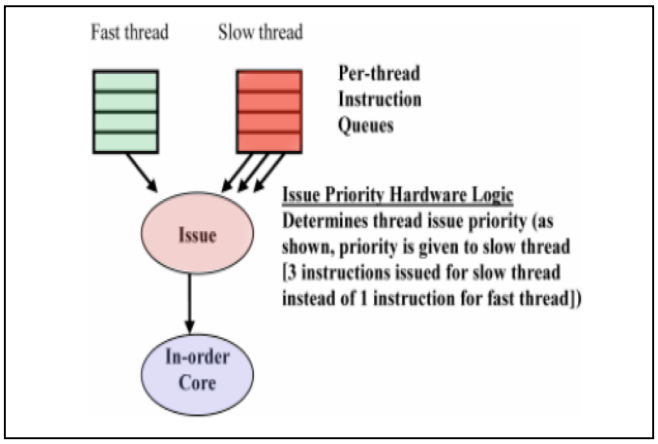

Figure 3: Thread Balancing

The Imbalance information is then used by the issue prioritization hardware logic which us used by the issue stage of the multithreaded processor.

C. **Energy Efficient Techniques**

1.Leakage Aware Largest Task First

This technique proposed by Jian-Jia Cen and Lothar Thiele is based on the idea that Dynamic power consumption due to switching activities and Static power consumption due to leakage currents are the two major sources of energy consumption of a CMOS processor. Dynamic voltage scaling is discussed in the previous section. Dynamic power consumption is directly propotional to the speed/frequency and this idea motivates to execute on lower speeds. Thus as discussed in the previous section power is scaled down by scaling down the voltage/frequency. This paper suggests that the critical speed which is the speed that the processor is turned down to to make it dormant when there is no job for execution might be too optimistic and that the processors dormant mode might be energy-inefficient. Instead of that, this paper proposes a new approximation algortihm which can reduce the energy consumption by 15%. This is because the algorithm from the previous section assumes that there is nooverhead involved for turning the processors to the dormant state. This can be explained as follows: The critical speeds are considered to be the limitation for task assignment but for m identical tasks under tight conditions ,the actual consumption according to the paper is amost 1.667 times the optimal solution. Thus, proposes a algorithm for the Leakage Aware Multiprocessor energy efficient scheduling problem.

Leakage-Aware Largest first is used to verify whether it is necessary to consider speed independent power consumption. This algorithm assigns the largest unassigned task, which is the defined as the task with the largest ratio of the amount of the required computational cycles to its period as describes in the previous section, to the processor with the smallest workload. If there are n tasks, tasks are indexed from the largest to the smallest arbitarily.The first *m* tasks are assigned to the first m processor. If after task *t*, if computation cycles to period ratio is greater than the critical speed and computational cycles and in the next iteration the ratio is less than the critical speed, It means that there exists a lower bounded solution which executes task till that iteration where the ratio is greater than the optimal speed which is now the computation cycles to the period ratio till the point where the ratio is greater than the critical speed. Then we schedule tasks to the *m-k* processors. Consider a set T of independent tasks over M identi- cal processors with a common power consumption function $P(s)=s3+\beta$,where $\beta \geqslant 0$,and all tasks in T are ready at time 0. Each periodic task $\tau i \in T$ is associated with a computation requirement in *ci* CPU-cycles and a period *pi*, where the relative deadline of τi is *pi.*The algorithm can be summarised as shown:

ALGORITHM :

Input: (T, M);

1. Sort all tasks in T in a non-increasing order ci/pi for τ ,i$\in$T;
2. Derive the critical speed s ;
3. Set l1,l2,$\cdots$,lM to 0, and T1,T2,$\cdots$,TM to $\varnothing$;
4. fori=1to|T|do
5. Find the smallest lm; (break ties arbitrarily with $1 \leqslant m \leqslant M$)
6. Tm $\leftarrow$ Tm $\cup$ { τ i} and lm $\leftarrow$ lm + ci ; Sort all the tasks in a non increasing order of *ci/ pi.*
7. Return the schedule SCLA+LTF which turns a processor into the dormant mode instantly when it is idle, and executes all of the tasks in Tm ($1 \leqslant m \leqslant M$) in an earliest-deadline-first order on the m-th processor at speed s0 if lm $\leqslant$ s0, or at speed lm, otherwise;

The tasks in T are assumed to be reindexed from the largest to the smallest, and the workload can be distributed to all the m processors in the critical speed.

3. Speed Scaling on Parallel Processors

This technique introduced by Sussane A, Fabian M and Swen S, investigates algortihmic instruments leading to lo power consumption. The paper claims that there are two mechanisms that can save energy on a logarithmic level. *Speed Scaling* and *Sleep States*. The first approach is that processors can be made to operate at variable speeds. The key being that higher the speed higher the power consumption owing to the cubic relationship between power and frequency. The other approach is to put the system to sleep state which is a low power idle state. The main problem with this approach is that it is difficult to find out when to shut down a processor since the transisiton back requires extra energy.

Polynomial Time Algorithm

The Polynomial time algorithm described in a previous work by Yao et al is a Round Robin algorithm. If we were given *n* jobs on *m* identical variable speed processors, specified by the release date *r(i)*, deadline *d(i)* and processing volume *p(i)* . The jobs are sorted according to their order of their release. The algorithm is described as follows.

1. Initially The jobs are numbered in order of non decreasing release dates. If two jobs have the same release dates they are sorted in order of their non-decreasing deadlines
2. Once the Sorted jobs are computed in step 1, assign the jobs to processors in a round robin fashion.
3. For each processor, if there are a certain number of jobs assigned to it, the optimal service schedule is computed.

Theorem: *For a set of unit size jobs with agreeable deadlines, the algorithm computes an optimal service schedule.*

In accordance with the given theorem ,

FACT 1: *Given a feasible schedule for jobs with agreeable deadline, on any processor the assigned jobs can be reordered such that they are executed in order of increasing job index. This rendering does not cause a higher energy consumption.*

The paper then considers a unit size jobs with arbitary release datas and dealines and a new polynomial time algorithm is developed. The paper also shows that the problem of minimizing the total consumed energy is a difficult problem.

Classified Round-Robin

The algorithm divides the given jobs into classes and then, when assigning jobs to processors applies round robin strategy independently for each class. The classification is done such that a class contains all the jobs of the same density. Density is defined as $1/(d(i)-r(i))$. *d(i)* is the Deadline of the task and *r(i)* is the Release date. Each job has a processing requirement. The algorithm can be described as follows:

1. For each class C, First the jobs are sorted in non decreasing order of release date and then assigned to processors in a Round Robin fashion.
2. For each processor, given the jobs assigned to it, an optimal service schedule is computed.

LEMMA:*For any number of jobs, the energy of an optimal schedule on m processors is at least* $1/m^{\alpha-1}$ *times that of an optimal schedule on one processor.*

Suppose we are given a job with $r(i) = 0$, , for all *i.* The deadlines d(i) may take arbitary values. This strategy combines *earliest deadline* and *List scheduling* to assign jobs to processors. At any time, the load of the processor is supposed to be the sum of the *p(i)'s.*

Earliest Deadline List Scheduling Algorithm

1. The jobs are numbered in order of non-decreasing deadlines which is $d(1) \leq \ldots \leq d(n)$.
2. The jobs are considered one by one in the order compute in step 1. Each job is assigned to the processor that currently has the smallest load.
3. Given the jobs assigned to it, each processor computes the optimal speed sequence using the optimal offline algorithm for a single processor.

All of the above algorithms, Polynomial Time Algorithm, Classified Round Robin Algorithm, Earliest deadline algorithms can be modified so that they work in an online scenario meaning that the jobs arrive overtime. That is, a job *i* togather with its correspondng characteristics *d(i)* and *p(i)* becomes available at its release date *r(i)*. This means that the jobs must be assigned to a processor without the knowledge of future jobs arriving at times $t > r(i)$.

IV. Thread Assignment on Heterogenous processors

A. Static Techniques

The best static known assignment techniques are based on the assumption that the characteristics of the workloads are known as *a priori*. The workloads if not known beforehand, static techniques might not prove to be a good option especially if there is a large diversity.

1.Random Static Assignment

The random static assignment method does not need to know the priori knowledge of the the workload. As the name implies the threads are mapped completely in a random fashion to the processors. If the system has more threads than there are cores, unassigned threads will be given a processor as soon as one is available.

2.The Pseudo best Static Assignment

The pseudo best static assignment assumes the runtime characteristics of the workload is known beforehand. It assumes the presence of an oracle that provides the programs IPC and the number of instructions to be executed. This means that we check for the best static assignment of threads to the cores. Instead of finding the best assignment the most common means is to find the suboptimal assignment based on a heuristic.

3.Phase guided Thread to core.

The resource needs of the threads must be rightly or closely met. That is the goal of a good assignment technique. This tecnique automatically determines the mapping between threads and performance-asymmetric cores of a processor. The main idea that is exploited in this technique is the fact that programs exibit phase behaviour. We take code sections and classify them and group them so that a particular group exibits more or less similar phase behaviour.Thus the exibited characteristics of the different sections on different types of cores can be used for automating thread-to-core assisgnment at a lower runtime cost. With multiple target architectures the mapping becomes very difficult to map manually by the programmer. *Phase behaviour* means that when the program goes through different phases of execution that phases consistently show similar runtime characteristicscompared to other phases. Thus, this approach consists of two parts. An Offline program analysis that is supposed to identify the transistion points in a program and a

lightweight dynamic analysis that determines thread to core mapping. The phase transisiton point is a point in the program where runtime characteristics are likely to change. These may not be distinct but most likely.

Phase guided assignment

If we can classify a program's execution into code sections and group these sections into clusters such that call clusters exibit similar runtime characteristics, thread to core assignments for unanticipated cases that might involve varying architectures can also be involved. The Offline analysis performed identifies the phase transisition points. Each phase transisiton point is instrumented to insert a *phase mark* which is a small code fragment. A phase mark plays the key role of analysing the dynamic performance and makes core switching decisions.The results of the analysis determine the suitable core mapping for the phase type.

The Offline Analysis is performed as follows. The program is divided into Procedures and each procedure is again divided into basic blocks. A block is a section of code that has one entry point and one exit point. Each block is then classified into exactly one type. From these atributed intraprocedural control flow graphs for Procedues are created. The control flow graph is then partitioned into unique set of intervals using standard algorithms. For each Interval, we compute its dominant type by doing a depth-first traversal of the interval starting from the entry node. A sample run of the traversal is shown in the paper is as shown:

On reaching a control flow node with an outgoing backward edge, if the edge has not been previously traversed, the target node is calculated. The weights are heuristically decided and

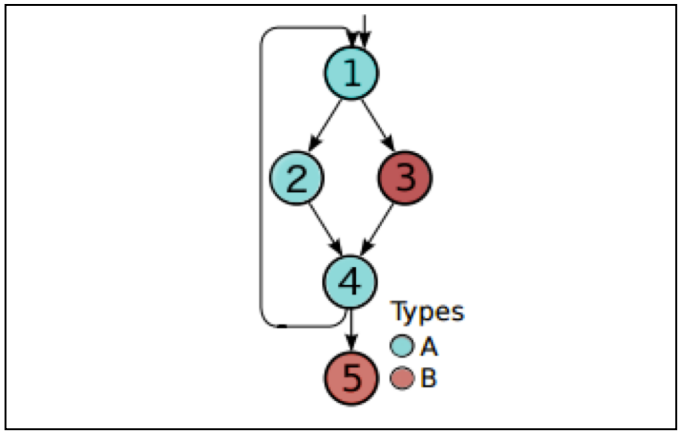

Figure 4: Offline Analysis Control Flow Graph

the Node weight function maps to the values based on heuristic measure of the expected time of the block and the dominant type of the interval is then decided.

During a depth-first traversal we maintain a stack of control flow nodes encountered thus far ($\rho = \eta + \rho'$) with the entry node of the interval at the bottom of this stack and the currently visited node at the top of the stack. A type map for the interval (M : $\Pi \rightarrow$ R) is maintained. On visiting a control flow node η in the interval, the type map M is changed to M' ,where M', is $M \oplus \{\pi \rightarrow M(\pi)+wf * \varphi(\eta)\}$. Here, π is the type of the control flow node, w_f is the forward edge weight, φ maps nodes to node weights, and $\oplus$ is the overriding operator for finite functions.

In Figure 3, 1 and 4 are the control nodes. The control nodes Map the control to the next node depending upon the weight of the edges which is given by the function M. After the Phase Transisiton analysis, *phase marks* are statistically inserted j_n the binary to produce a standalone binary with phase inforamtion and dynamic analysis. The attributed Control Flow Graph is then considered section by section, The approach is to mark all the edges in the Control Flow Graph where phase and target sections have different phases.

Thus, After the Phase transisiton marking is complete, we have a modified binary with *phase marks* in the Control Flow graph. The *phase marks* not only switches cores between transistions between phases but also monitors the current performance characteristics and determines the phase type if the phase type has not been determined previously.

Thus, Phase-Guided Thread to Core Assignment improves the utilization of Assymetric Multicore processors statically.

B. **Dynamic Techniques**

1. Round Robin

Threads can be assigned to cores in a round robin fashion just like in the Homogenous processors round robin assignment. That is to periodically rotate the assignments of the threads to the processors. This policy ensures that all cores are assigned threads and no core remains idle. A method defines the frequency of the rotation. The Round Robin strategy is blind. It is unaware of the runtime information and at that frequency that is decided, it continously allocates threads to the processors.

2. IPC Driven

Round Robin mechanism can be improved by considering the characteristics of the executing threads. A good assignment as the programs enter different phases. Relative IPC can be used as a metric to quantify thread behaviour and assign dynamically. The value of the IPC of the running threads can be easily made at the end of each execution cycle. The overall performance can be improved for a heterogenous architectures if threads are executed on the right cores meaning that the threads which meet their resource requirements. Threads can first run on some available core and migrate to the right core when the faster core becomes idle. This kind of thread migration although might seem useful carries with it a overhead and sometimes might prove costly.

In order to implement IPC driven thread assignments, First, the IPC values on all processors must be available in order to make assignment decisions. There should be some kind of a learning mechanism involved in order to learn the IPC values at the end of each cycle or at regular intervals. It must also be defined as to how often and how the control information can be or must be used. It is assumed that the program's current IPC is always available for the processor that is executing the program since the program executes only on one processor at a time.

The Key idea is to control the thread migrations in IPC Driven assignments. Forced migrations performed in order to keep the IPC estimate accurate. IPC estimate can be intialised in two ways.

1. Periodically on all threads at the same time
2. Per-Thread basis

Each program from the section on Phase-guided thread assignments has a unique IPC. The patterns differ in both shape and phase duration. Therefore, a single migration maynot be suitable for all programs. Long threads would penalise threads with shorter exectution time and short threads would trigger useless migrations which involve overheads and might not be very energy efficient.

The phase behaviour of the program does not depend upon the executing processor. By this fact, forced thread migrations can thus be triggered by a raoid variation in the IPC. The use of bare relative IPC as a control variable would cause continous variations of IPC within a limited range to trigger frequent migrations. In order to avoid this phenomenon, the IPC driven mechanism uses the moving average of the IPC as the control variable. Additionally a forced migration is triggered at the beginning of the system to initialize the system. This process also ensure that no processor remains idle by migrating unassigned threads to the all the processors like in the Round Robin case.

It is experimentally proven that dynamic assignments improve performance of the heterogenous mutlicore processors. The presence of may low area cores ensure a high level pf parallelism and the when thread parallelism is low, high performance cores ensure good throughput.

3. Dynamic Assignment for Intra-thread Diversity

This "Best Core assignment" strategy proposed by Rakesh K, Dean M. T, Parthasarathy R, Norma P. J and Keith I Farkas demostrates that dynamic core assignment policies that provide significant performance over naïve assignments and even outperforms the best static assignment. The heterogenity comes from different cache sizes, their raw execution bandwidths and other fundamental characters such as inorder and out of order processors. Other architectures also seen to address both single threaded latency and multiple threaded latency. Simulataneous multithreading can devote all processor resources on a superscalar processor to a single thread or divide them among several.

Workload-to-core mapping is one dimensional problem as the workload consists of a single running thread. With multiple cores and multiple threads, the task is not to find the best core for the thread but to find the global best assignment. The paper strives to maximize average performance gain over all applications in workload. Fairness is not taken into conderation explicitly. There is another problem with the concurrently executing threads which is Cache Conherence. This is overcome in this method by using disjoint address spaces. It should be clear by now that there are more advantages when exploiting core diversity for inter-workload variation.The paper eexamines implementable heuristics that dynamically adjust the mapping to improve performance. These hueristics are sampling based. During the execution of the workload there is a trigger that initialises the sampling phase. In the samplig phase, the scheduler permutes the assignment of appication to cores, changing the cores onto which the applications are assigned. The dynamic execution profiles of the applications being run are gathered by the referencing hardware performance counters. These profiles are then used to create a new assingment which is then used for a much longer phase called the steady state and so on. The steady state continues until a next trigger is trigger and new assignments are made. The main part that implements such a strategy is the sampling mechanism. This method focuses on the core sampling strategies.

The first Strategy is called *Sample-one* which samples as many assignments as is needed to run each thread once on each core-type. This assumes that the single sample is accureate of what other jobs are doing. Then the assignment is made, maximising weighted speed up under another assumption that the future performance will be the same as the single sample for each thread.

The Second Strategy , called the *Sample-avg* assumes we need multiple samples to get the average behaviour of a job on each core. We run the threads atleast two times and the sampling is done as many times the threads are run. The assignment then is based on the average performance of each thread on each core. This assignment is more consistent thread migration costs are less.

The Third Strategy called the *Sample-sched* assumes we know little about the particular assignment unless the thread is actually run. Hence, it samples a number of possible assignments and then is constrained to choose one of the the assignments that it has sampled. Selection of the best core of those sampled is the one that maximises total weighted speedup. It is experimentally proven that the sample-sched approach gives the best performance.

There is also the issue of how to trigger the assignments, Sampling effectively requires reacting quickly to changes in the workload and also minimising the sampling overhead. To manage this tradeoff between the senstitivity to changes in the workload and the effectiveness of the samepling, Two Trigger Mechanisms are proposed based on a Periodic Timer and the other based on events indicating significant changes in performance.

Time triggered sampling is simple to implement and , it does not capture either inter thread diversity of intra thread diversity. Fixed sampling frequency may not be adequate when the phase lengths of different applications in the workload mix are different. Each application can demonstrate multiple phases with different phase lengths and hence time triggered sampling may not be useful for all applications.

The Second Class of Trigger Mechanisms monitor the run-time behaviour of the workload and detect significant changes. Three instantiations of this trigger class are considered.

Individual-event trigger is triggered every time the steady state IPC of an individual thread changes by more than 50%.

Global-event trigger sums the absolute values of the percentage changes in IPC for each application and the trigger is triggered when the sum is 100%.

Bounded-global-event trigger modifies the global-event trigger by initialting the a sampling phase if more than 300 million cycles has elapsed since the last sampling phase. In other words it samples every 300 cycles if there is no sampling that takes place and avoiding triggering if the last sampling has been triggered not more than 50 cycles ago.

The paper experiments with the three mechanisms and concludes that the *Bounded-global-event trigger* performs better than the others.

Thus dynamic assignment is possible by sampling performance overtime and triggering the sampling at the right time.

4. Adaptive Mapping

The paper written by Chi-Keung Luk, Sunpyo Hong and Hyesoon Kim introduces a heterogenous programming system called *Qilin. "*In order for the software to fully realize this potential, the step that maps the computation to processing elements must be as automated as possible" According to the paper. The general approach is to rely on the programmer to specify this mapping manually and statically. This approach is difficult and labor intensive. The method that is proposed called *Adaptive Mapping*, a fully automatic technique to map computations to processing elements on CPU+GPU. In order for mainstream programmer to effectively work on the heterogenous architecutures is to make the mapping of the computations to the blocks must be as automated as possible. Current CPU+GPU computation mapping. Qilin is based on the Intel Threading building block(TBB) and The Nvidia CUDA for the GPU. Instead of directly generating CPU and GPU native machine codes, the Qilin compiler generates TBB and CUDA source codes from Qilin programs on the fly. Qilin dynamic compilation consists of four steps listed

1.Building Directed Acyclic Graph: DAG's are build according to the data dependencies. These DAG's are essentially the intermediate representation which later steps to compilation process.

2.Deciding th Mappings from the Computations to processing elements: This step uses automatic adaptive mapping to decide the mapping.

3.Performing optimizations on DAG's:Operation Coalescing and removal of unnecessary temporary array are some of the optimizations that can be performed in this step.

4. Code is Generated after the optimizations are performed once all resource constraints are taken care of. Qilin also generates all the gluing codes that are needed to combine results from the CPU or the GPU.

The Actual adaptive mapping mechanism works as follows:

The Qilin adaptive mapping automatically finds the nearest optimal mapping from computations to processing elements for the given application. A program is run on both the CPU and the GPU.

Tc = Actual time of execution on the CPU.

Tg is the actual time to execute the given program on the GPU.

Tc' = the Qilin projection of Tc

Tg' = the Qilin projection of Tg

Tc and Tg are predicted by using a analytical table based on static analysis. While this approach might not work very well for complex programs. Qilin takes an empirical approach. Qilin maintains a database that provides execution time projections for all programs it has ever executed. The first time a program is run is called a *training run.* Suppose the input problem size is known, Qilin divides it into two parts. One part is mapped to the CPU and the other part is mapped to the GPU. Within the CPU it further divides the subpart into smaller sibparts and the execution time is measured. The same is followed for the GPU. Once all the execution times on CPU and GPU's are available, The projections for the actual execution times is constructed as a linear equations using curve fitting. The next time a different program is run, it uses the projections for the actual execution times , it uses the projection values to determine the computaion to processor mapping.

Thus, the Adaptive Mapping is an automatic technique to map the computations to core on heterogenous multiprocessors.

2.HASS: A scheduler for Heterogenous Multicore processors

The paper written by Daniel S Juan Carlos S. A and Stacy J claims that single ISA multicore processors will have an edge in potential performance per watt over comparable homogenous processors. The OS scheduler needs to be aware of the heterogenity of the processors. The paper proposes a hetrogenity aware signature supported scheduling algorithm that does the matching of the tasks to the core. Static nature of HASS imposes some ,limitations on its structure and functionality.

To achieve this goal, the main thing that should be given focus is the architecural signatures. HASS relies on the abilty to estimate potential performance of a thread on a core given the load characteristics of that core.

Constructing Signatures:An important propery of signatures is their microarchitecture independence. The signature should be available to the OS at the scheduling time. So the ideal place to hold is the application binary itself. In order to construct the signature we need the reuse distance profile which is collected via offline profiling. All that needs to be done is execute the program with the profiler that will generate the signature and embed it into the binary. Several runs can be performed and the results can be combined into one signature. Once the profile is collected cache misses is estimated for realistic cache configurations. These estimations collected in a matrix comprises the architectural signature.

Using Created Signatures for Scheduling: At runtime the architectural signature is used to estimate the threads performance on each type of core. To accomplish this, two separate parts are considered called the Execution time and the stall time. Executing time is the amount of time it takes to execute the instructions assuming a constant number of cycles oer instruction. Constant memory latency is assumed and no uniforam memory access latency may cause inaccuracy. The

resultant sum of both time compnents gives us an abstract completion time metric.

The method also reflects on shared caches, the performance of the shared caches is not only affected by the frequency of the core and the properties of the application but also by the cache access patterms. The performance of different threads on differenct cores based on cores' cache size and frequency. This allows threads to distinguish Cores by their realtive desirabilty.

HASS scheduler: HASS scheduler stands for Heterogeneity-Aware Signature Supported. The algorithm also emphasizes scalabilty to accommodate more cores in the future. The first abstraction made is called the *processor class*. Each processor is said to be in one class based on the features such as clock frequency, cache hieracrhy. If two cores belong to the same class they must be identical. To be aheterogenous processor, the system must have atleast two such classes, Each class or partition keeps count of the runnable threads. Class and pratition are no the same. If there are a large number of cores it becomes hard to classify them into classes so the solution is t group them in partitions in the CPU. Each class may have more than one partition but wach partition must have the same class.

When threads enter the system, they iterate through all the existing processor classes and estimate the performance using signatures according to the attributes of the class. To assign itself to a partition, the thread goes through the list of all the partitions using the base rating and current number of runnable threads and selects the partition with the highest expected performance and assigns itself to it. This process is called *Regular assignment.* When the number of threads in the partition changes it is called a *refresh.* There is no load balancing between partitions and more powerful partitions may get higher loads and some partitions may get underloaded. This is prevented by forbiding to move to fully loaded partition when there is a underloaded partition available. The greedy approach has a potential problem where threads may becomes locked in a suboptimal assignment and there is a need for awap. This is achieved by *optimistic assignment.* The intiator thread can only trigger the switch if it sees that the swap will increase the overall performance. The search for a partner can be slow when there are lots of partitions.

The partition scheme allows the scheduler to avoid global lock during scheduling. Instead , threads can lock one partition at a time during the *refresh*.

The paper claims that IPC-driven algorithm revealed that IPC ratios were often inaccurate due to unstable nature of phase changes. HASS is a novelty in using offline generated architectural signatures for determining the threads assignments in assymetric multicore processors.

C. **Energy Efficient Techniques**

1. Energy Metrics based Power Reductions.

This paper written by Rakesh K Keith I. and Norman P, Jouppi, Parthasarathy R and Dean M T proposes a single-ISA heterogenous multicore architecure as a mechanism to reduce power consumption. As processors continue to increase in performance and speed, processor power consumption and dissipation are major challenges. This method tries to reduce processor power dissipation. The architectures are mainly chip-level multiprocessors and with multiple diverse cores. These cores all execute the same instruction set but have different perfoemance capabilities. The goals a previously discussed could be power efficiency or execution speed ups. There are many reasons why the best core of execution may change over time. The demands of executing code may vary widely between applications . The best core for a particular thread may not be the best core for some other thread. Even s single application may have multiple phases that may require different resources for each phase. Although not all of these factors are considered in any heuristic, this method specifically examines adaptation to changes in the phases.

There is a cost to Switching cores, so we must do it reasonably. This could either mean that we must restrict switching only to operating system timeslice ontervals, with user state already saved to memory and new core would then power down the old core and return from the timer interrupt handler.

The Oracle Heuristic for dynamic core selection depend on the particular goals of the architecure or application. The Oracle Algorithm maximize two sample functions. The first optimizes for energy efficiency. The second optimizes for energy-delay product with a looser performance constraint. Choosing the Core that minimizes energy or the energy-delay delay product over each interval subject to performance constraints although produce good results, it does not give an optimal solution for the global energy or energy delay product.

Oracle based on Energy Metric

This seeks to minimize the energy per commited instruction and thus energy used by the entire program. For each interval, the oracle chooses the core that has the lowest energy consumption under the constraint that the performance has to be always maintained within 10% of the maximum performance.

Oracle based on Energy-Delay metric

The second Oracle utilises the energy-delay metric. The product seeks to characterize the importance of both the energy and the response time in a single metric. Under the assumption that they have equal importance. This is achieved by minimising the energy-delay product by always selecting the core that maximises IPS^2/Watt. The performance constraint being that each core should maintains performance within 50% of the maximum performing processor.This is different from Chip-wide Voltage /Frequency scaling.

This paper also goes on to prove that dynamic core selection mechanisms give more performance than static techniques.

Thus two energy efficient techniques are introduced based on Energy metric and Energy-Delay Metrics are examined and this widens the scope of utilising these Metrics for much Energy-Efficiency.

2.Regression Model

This method proposed by Jason Cong and Bo Yuan proposes a energy efficient scheduling methos as follows:

1. A regression Model is developed to estimate the energy consumption on Intel's QuickIA heterogenous prototype platform

2. An Energy-efficient scheduling approach is proposed to map the program to eht most approporate core based on th program phases using combinations of static analysis and runtime scheduling.

The metric that is used to characterize the energy efficienty is the energy delay product over instruction intervals. The method uses *Regression model* to predict energy consumption. Training data to train the model is developed to train the model. Some benchmarks are run on the hardware and each time, fifteen pieces of hardware performance is collected. With the training data samples. The training data is we build and evaluate the regression model. After identifying the hardware performance parameters that are Cycles, Retired Instructions, L1 D cache access and L2 D cache access, The total energy sonsumed is calculated. Based on the Ordinary Least squared methods the regression coefficients are determined.

The Program phases are identified in the same way as in the Phase-guided threads assignment technique by using a call graph.

The scheduling decision is made at runtime using the instrumented codes. The correct energy delay products and the regression model needs the four key hardware performance parameters to accurately predict the energy consumption.

Thus, In the paper an energy efficient scheduling method for heterogenous multicore architectures is proposed . Regression model is used to estimate the energy consumption.

V. CONCLUSION

The survey gives the idea that thread assignment policies although many, may finally be classified into 3 functional categories depending upon how they operate. The first Category tries to implement a thread guiding mechanism based on the Phases of the program and the change in the phases or simply analysing the code offline. This is done by a mechanism that learns and categorizes the threads to be assigned accordingly. The other main goal that is achieved which is power conservation is either done by decreasing the Voltage/frequency of the core called *Voltage Scaling* or the processor thus conserving energy or by offline analysis of the energy consumption of the code or energy metrics such as energy delay metric dynamcally. The offline analysis is either based on a profiler or a learning mechanism that learns the phases or analyses the program determines the workload metrics and assignments are made accordingly.

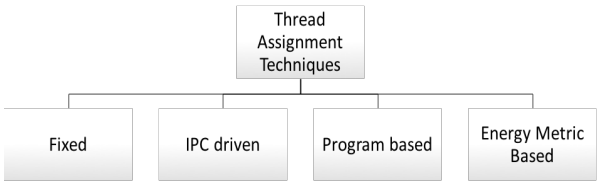

Figure 5: Simplified Taxonomy

The Performance which is either to improve the throughput and the Energy Conservation per core or for the whole CMP is achieved by

1. Core Switching
   Threads migrate from one core to another and the efficient thread migration techniques are used to switch between cores depending upon the resource demands of the thread.

2. Voltage Scaling: Voltage/frequency is scaled down to run a thread for much longer time and hence make all the cores finish at the same time making sure no core is idle.

3. Core Dormant States: If there is no workload assigned to a core, the core is put into dormant state and is switched between dormant and active states accordingly.

ACKNOWLEDGMENT

I thank Professor Soner Onder, from the bottom of my heart for helping me throughout the process of writing this paper, for complete walk through of how the paper needs to be written, Even for sparing time out of his busy schedule to review and help draft the paper. I sincerely thank professor Onder for this wonderful learning experience. I feel privileged to have sat in his class as a student.

REFERENCES

[1] Rakesh Kumar, Keith I. Farkas, Norman P. Jouppi, Parthasarathy Ranganathan, and Dean M. Tullsen. 2003. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. In Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture (MICRO 36). IEEE Computer Society, Washington, DC, USA, 81-.

[2] Rakesh Kumar, Dean M. Tullsen, Parthasarathy Ranganathan, Norman P. Jouppi, and Keith I. Farkas. 2004. Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance. In Proceedings of the 31st annual international symposium on Computer architecture (ISCA '04). IEEE Computer Society, Washington, DC, USA, 64-.

[3] Daniel Shelepov, Juan Carlos Saez Alcaide, Stacey Jeffery, Alexandra Fedorova, Nestor Perez, Zhi Feng Huang, Sergey Blagodurov, and Viren Kumar. 2009. HASS: a scheduler for heterogeneous multicore systems. SIGOPS Oper. Syst. Rev. 43, 2 (April 2009), 66-75. DOI=http://dx.doi.org/10.1145/1531793.1531804

[4] Tyler Sondag and Hridesh Rajan. 2009. Phase-guided thread-to-core assignment for improved utilization of performance-asymmetric multi-core processors. In Proceedings of the 2009 ICSE Workshop on Multicore Software Engineering (IWMSE '09). IEEE Computer Society, Washington, DC, USA, 73-80. DOI=http://dx.doi.org/10.1109/IWMSE.2009.

[5] R. Rakvic, Q. Cai, J. González, G. Magklis, P. Chaparro, and A. González. 2010. Thread-management techniques to maximize efficiency in multicore and simultaneous multithreaded microprocessors. ACM Trans. Archit. Code Optim. 7, 2, Article 9 (October 2010), 25 pages. DOI=http://dx.doi.org/10.1145/1839667.

[6] Michela Becchi and Patrick Crowley. 2006. Dynamic thread assignment on heterogeneous multiprocessor architectures. In Proceedings of the 3rd conference on Computing frontiers (CF '06). ACM, New York, NY, USA, 29-40. DOI=http://dx.doi.org/10.1145/1128022.1128029

[7] Jian-Jia Chen and Lothar Thiele. 2010. Energy-efficient scheduling on homogeneous multiprocessor platforms. In Proceedings of the 2010 ACM Symposium on Applied Computing (SAC '10). ACM, New York, NY, USA, 542-549. DOI=http://dx.doi.org/10.1145/1774088.1774198

[8] Susanne Albers, Fabian Müller, and Swen Schmelzer. 2007. Speed scaling on parallel processors. In Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures (SPAA '07). ACM, New York, NY, USA, 289-298. DOI=http://dx.doi.org/10.1145/1248377.1248424

[9] Jason Cong and Bo Yuan. 2012. Energy-efficient scheduling on heterogeneous multi-core architectures. In Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design (ISLPED '12). ACM, New York, NY, USA, 345-350. DOI=http://dx.doi.org/10.1145/2333660.2333737

[10] Chi-Keung Luk, Sunpyo Hong, and Hyesoon Kim. 2009. Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 42). ACM, New York, NY, USA, 45-55. DOI=http://dx.doi.org/10.1145/1669112.1669121

[11] Robert I. Davis, Alan Burns, Jose Marinho, Vincent Nelis, Stefan M. Petters, and Marko Bertogna. 2015. Global and Partitioned Multiprocessor Fixed Priority Scheduling with Deferred Preemption. ACM Trans. Embed. Comput. Syst. 14, 3, Article 47 (April 2015), 28 pages. DOI=http://dx.doi.org/10.1145/2739954