Abstract—Thermal hot spot and high temperature gradient degrades the reliability and performance of chip multiprocessor. This is an important issue in the current days high transistor density chip multiprocessor. In this paper, we explored the benefits of different temperature aware scheduling and mapping approaches of applications on chip multiprocessor to reduce the peak temperature. As most application’s run time exhibit phase wise behavior, we have exploited the run time phase wise power consumption behavior of the applications to schedule and map the applications to multicore chip to reduce peak temperature. We have evaluated five scheduling approaches (critical path, modified critical path, energy capped critical path, naive load balancing, and task partitioning and scheduling (TPS)) and five mapping approaches (random, greedy, row-col, checker board and boundary fix checker board) for both synthetic data and real benchmarks on assumed 8 × 8 chip multiprocessor. We have taken benefit of both (a) optimal scheduling of tree or chain of unit time tasks on multiprocessor using critical path heuristics and (b) phase wise behavior of applications. Result shows that greedy based mapping approach performs better compared to simple low overhead (without increasing the execution time in executing phase wise applications on chip multiprocessor).

I. INTRODUCTION

Modern computing system contains multiple cores which enable many applications or tasks to execute concurrently. Chip multiprocessor exploits the increasing device density in a single chip, so now a days we expect two or three order number of cores on a single chip. As the number of processors increase in the chip, the power consumption per unit area increases, and also the temperature of the chip. If temperature of the chip increases above the allowable value, some transient faults may occurs in the chip and chip is not reliable above that temperature. It also introduce permanent fault if the peak temperature rise above some threshold value. So for reliable operation of the chip we need to maintain the peak temperature of the chip below the maximum allowable temperature.

Most of the application’s run time characteristics exhibit time varying phase behavior [1]. Applications impose different values of performance metrics in different phases of the application’s execution. During execution of a phase an application the value of performance metrics remain same. In [2], Banerjee et al. used instruction per cycle (IPC), instruction level parallelism (ILP) and L1 cache hits to detects phases of applications execution time. Table I, shows phase wise behavior of SPECINT, SPECPF and media benchmarks [2].

In this work, we have considered the phase wise power consumption characteristics of multiphase applications to efficiently schedule the applications onto chip multiprocessor to reduce peak temperature of the multicore chip. Without loss of generality, we can consider an application consists of a sequence of phases or tasks, where each phase (or task) exhibit different power consumption characteristics. In this paper, we are concerned about scheduling of N multiphase applications (chain of tasks, each of which have arbitrary number of phase or task) onto chip multiprocessor with M processors to reduce peak temperature of the chip. Each phase or task has two characteristics: execution time of the phase and power consumption of that phase. A phase or task of an application can be scheduled on any processor, irrespective of processor number the task consume same amount of power. If execution time of a task is T then we say it is a unit time task and for unit time task we do not use pre-exemption. In this text, we have used the terms task and phase interchangeably.

In this paper, we evaluated five scheduling approaches and five approaches of mapping the scheduled task on to chip multiprocessor to minimize execution time and peak temperature of the chip. The performance of min-min (the greedy approach) approach and the state of art task partitioning and scheduling (TPS) are also considered [3], [4]. TPS approach uses the time slot wise interruptive scheduling and min-min task mapping uses temperature sensor (or predicted temperature) data. But the performance of both reducing peak temperature and overall execution time is not as good as compared to the basic critical path based scheduling in combination with simple location based mapping for multiphase applications on to chip multiprocessor.

The rest of the paper is organized as follows: We have described previous work related to thermal model, monitoring, design and management (scheduling and mapping) to reduce peak temperature of chip in Section II. In Section III, we have described formal model of multiphase application that is phases wise power consumption behavior, and also described about the thermal prediction model of chip multiprocessor based on thermal resistance and capacitance (RC) model and matrix values. We have described our used five scheduling approaches to reduce the peak temperature of the chip with or without increasing the overall makespan of the task system in Section IV. We have described our used five mapping approaches of the scheduled tasks to reduce overall peak temperature of the chip and compared the performance with state of art approach in Section V. Experimental evaluation and analysis of result for the scheduling and mapping policies are given in Section VI. Finally, we have concluded about our work and discussed about possible future extensions to the work.

II. PREVIOUS WORK

We can classify previous work related to thermal aware mapping or management of task to multi-core system in two main categories. These categories are (a) thermal monitoring and modeling of chip multiprocessor and (b) thermal aware scheduling and mapping of task onto chip multiprocessor. In Kong et al. [6], they have given a detailed survey of recent techniques for temperature aware thermal monitoring, modeling, mapping and management in multiprocessor chip. Also in [7], [8], Kudithipudi et al. presented a general overview of thermal management in many core systems, and progress and challenges of temperature-aware computing.

Broadly recent works on thermal monitor, thermal aware design and thermal modeling of chip multiprocessor are reported in [9], [10], [11], [12], [13], [14] and [15].

In Skadron et al. [9], they described temperature aware micro-architecture modeling and implementation of single core CPU chip by using thermal RC (thermal resistance-thermal capacitance) model. In Cuesta et al. [11], they have developed 3D thermal-aware floor planner using multi objective evolutionary algorithm to reduce the peak temperature of their targeted Intel x86 many core chip. Long et al. [10] described an efficient thermal monitoring mechanisms for chip multiprocessors by placing sensors and interpolating those sensor data to find hot spot in both single core chip and chip multiprocessor. And in [12], Zhang et al. described a method for accurate temperature estimation using noisy thermal sensors to support dynamic thermal management.
A. Multiphase Application

In this paper, we consider a set of $N$ applications $S = \{A_1, A_2, ..., A_N\}$ to be executed on $M$ identical processors of a chip multiprocessor with minimum overall execution time and minimum peak temperature of the chip. Application $A_i$ has $n_i$ phases or tasks, where $i \in [1, 2, ..., N]$. Each task has two parameters: one is execution time and other is power consumption of the task. Task $Task_{ij}$ is $j^{th}$ task (or phase) of $i^{th}$ application $A_i$. $P_{ij}$ and $t_{ij}$ are power consumption and execution time of task $Task_{ij}$ respectively. Values of $t_{ij}$ and $P_{ij}$ may be arbitrary. Figure 2(a) shows an example of an application system. In this example, we have 4 applications $A_1, A_2, A_3$, and $A_4$. Application $A_1, A_2, A_3$ and $A_4$ have 4, 3, 5 and 4 phases, respectively. A task (or phase) of an application can not start execution before complete execution of its predecessor task (or phase) of the same application.

Energy consumption of a task (or phase) is product of execution time of the phase and power consumption of that phase, and is given by:

$$E_{ij} = P_{ij} \times t_{ij}$$  

Without loss of generality, we can convert the task $Task_{ij}$ in to a chain of $t_{ij}$ number of unit time task with power consumption $P_{ij}$. Figure 2(b) shows an example of the conversion of chain of tasks with arbitrary execution time to chain of tasks with unit execution time. When task execution time is one unit, we can use the energy consumption and power consumption interchangeably. Figure 1 shows, power consumption of different benchmarks (namely mc1, apsi and ibm) for first 500 time slots (each time slot is of 10,000 cycles). It shows that, the power consumption of benchmark varies from 2 watts to 20 watts in the different time slots. Mostly used optimization criteria of multiprocessor scheduling is minimizing makespan time $C_{max}$. The makespan is defined as the total length of the schedule i.e. when all tasks of all the applications have finished their execution i.e.

$$C_{max} = \max \{F_i\}$$

where $F_i$ is the finishing time of $i^{th}$ application.

In this paper, we also assume that there is no communication delay between tasks of an application and among the applications. In the paper, we assumed $(a) n = \sum_{i=1}^{N} n_i$ as total number of tasks (which is different from $n$, the number of phases of application $A_i$), (b) $m$ or $M$ as number of processors in the system and (c) all the applications arrive to the system at time 0.

To compare our work with state of art task partitioning and scheduling (TPS) method [3], [4], let us define a task as hot task if the task energy consumption value is above the average energy consumption, otherwise it is a cool task. The total energy consumption of the task system is

$$E_{total} = \sum_{i=1}^{N} \sum_{j=1}^{n_i} E_{ij}$$

and the average energy consumption of the task system

$$E_{avg} = \frac{E_{total}}{n} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{n_i} E_{ij}}{N \sum_{i=1}^{N} n_i} = \frac{\sum_{i=1}^{N} \sum_{j=1}^{n_i} E_{ij}}{N}$$

Based on this two term, we can easily classify a task as hot task or cool task. Task with energy consumption value above the $E_{avg}$ can be marked as hot task, otherwise cool task. We have also used the term $E_{avg, i} = \sum_{j=1}^{n_i} E_{ij}/n_i$ (that is average power consumption of application) to mark an application as hot application or cool application based on the $E_{avg}$.

B. Thermal Model of Chip Multiprocessor

We considered thermal resistance-capacitance (RC) model to express the underlying thermal process of chip multiprocessor as described in [15], [24], [9]. If the average power of core is $P$ over a time period $t$, then temperature $T(t)$ at the end of this period can be calculated as:

$$T(t) = P \times R_t + T_A - (P \times R_t + T_A - T_i) e^{-t/R_t C_t}$$

where $R_t$ is the thermal resistance and $C_t$ is the thermal capacitance, $T_A$ is the ambient temperature and $T_i$ is the initial temperature. By ignoring the nonlinear factors of equation 5, we can express the steady state temperature $T_{steadyState}$ as follows:

$$T_{steadyState} = P \times R_t + T_A$$

Based on equation 5 and equation 6, given the same value of $R_t$ and power profile $P$, the steady state temperature $T_{steadyState}$ is either
the lowest or highest temperature $T(t)$ that can be reached in the steady state. In particular, this should work well where the average time requirements of execution of tasks is large enough that a state close to the steady state can be reached (in this paper, we used 10,000 cycle as time period). The steady state temperature should generally serve as an upper bound on the actual maximum temperature, and that can be realized using simple matrix model for temperature of on chip multiprocessor as mentioned [15]. This can be described as follows

$$T_{k,j} = T_{k,j} + \sum_{x \in Adj_{j}} A_{x,j} (T_{k,x} - T_{k,j}) + B_j P_{k,j} + C_j (T_A - T_{k,j}) \forall k,j$$

(7)

$$T_{k,j} = T_{init}, \forall j, \text{ and } T_{peak} \geq T_{k,j}, \forall k,j$$

(8)

Where $T_{peak}$, $T_{init}$ is the initial temperature, core $j$ will execute task $i$ on time slot $k$, $T_{k,j}$ is the temperature of core $j$ at time slot $k$, $x$, is a neighbor of core $j$ and $Adj_{j}$ is set of all the neighbors of $j^{th}$ core. Value of matrix $A$, vector $B$ and $C$ are based on the thermal characteristics of the chip [15]. First, second, third and fourth terms of the equation 7 represent the effect of the temperature of the self processor, the effect of temperature of the neighbors, power consumption of the running task on the processor and effect on the ambient temperature respectively.

In [15], [10], authors have given a procedure to calculate the values of matrix $A$, vector $B$ and $C$ for a specific chip by mapping predefined kernel of tasks. They require to sense the temperature from all the processors of the chip and iteratively update the value of $A$, $B$ and $C$ to represent the thermal parameter of the specific multicore chip. This kind of prediction model helps to schedule and map the application to reduce the peak temperature of the multicore chip. In general, every processor has 3 to 7 neighbors in chip multiprocessor and it depend on the chip layout. But we can safely assumed that Row x Col processors connected in a grid, and each processor have four neighbors namely up, down, left and right to the processor. When the chip feature size is small, the effect of matrix $A$ is higher, that is effect of the temperature of the neighbor processors. We have used the neighbor effect term as collective parameter for the matrix $A$. Thermal monitoring mechanism for chip multiprocessor and accuracy in measurement and modeling of thermal behavior are given in Long et al. [10] in detail.

IV. SCHEDULING MULTIPHASE APPLICATIONS TO MULTIPROCESSOR

Scheduling defines the respective execution time slots for all the tasks in which the tasks get executed, and mapping decides the locations (on which processor) of the task execution at the scheduled time. In our case, we have used scheduling and mapping separately.

A. Hu’s Approach (HU): Scheduling tasks to minimize execution time

As shown in Algorithm 1, we have used the highest level first (HLF) to get optimal makespan (minimum makespan). This algorithm uses Hu’s [25], [26] highest level heuristics approach. The highest level (or the same critical path) heuristics to schedule tree (can be used for chain) of unit time task on multiprocessor is proved to be optimal and produce minimum execution time (schedule length). Here level of a task is number of tasks of the application following the same task, and it is same as remaining tasks of the application. Level of last task of an application is zero and level of tasks increase as we move from last task to first task of application. So level of task $Task_{k,j}$ is equal to $n_t - j$, where $n_t$ number of task of the application $A$.

B. Modified Hu’s Approach (ModHU): schedule mix of hot tasks and cool tasks as much as possible in the current time slot without increasing schedule length

As Hu’s algorithm do not consider energy consumption values of tasks, it generally produce a schedule with unbalanced amount of total energy consumption at different time slots of the schedule. So in the modified Hu’s approach, heuristically, we try to solve by choosing balanced amount of hot tasks and cool tasks in the current time slot if possible without increasing the schedule length.

Immediate solution for this problem is to wisely choose proper mix of both hot tasks and cool tasks if there are many candidate options (with the same level) for the same time slot. This will possibly balance the energy consumption in the current time slot and future time slots. As shown in Figure 3, at every time slot we are trying to choose one hot task and one cool task. This balance the total energy consumption in the different time slots and hence in term may reduce the overall peak temperature by allowing better mapping on to chip multiprocessor. Duty of the mapper is to map the hot and cool task to processor (in time: consecutive time slots and in space: neighbor processor) alternately if possible.

Suppose for a time slot $T_s$, the number of ready task is $k$ with same current highest level and $k$ is greater than the number of available processor $n_k$. In this case, we choose a set of $m$ tasks from $k$ ready tasks by making a proper mix of hot tasks and cool tasks. In our case we sort all the ready tasks based on their $EC$ values (energy consumption of the task), and choose $m/2$ hot tasks (higher $EC$s) and $m/2$ cool tasks (lower $EC$s) to make a proper mix for the time slot $T_s$. This kind of mix may provide a good balance of total energy consumption in the current time slot and also in the future time slots and hence allow to reduce the peak temperature of the chip.

Other solution is if there are time slots where some processors are not able to get unitized fully, then we can reschedule some of the tasks to later time slot without increasing the makespan time. As shown in Figure 3, we can exchange tasks of time slot $T_5$ and $T_6$ without increasing the makespan time. In the time slot $T_5$ two hot tasks are scheduled and in time slot $T_6$ two cold tasks are scheduled, by exchanging time slot of task ‘f’ and task ‘a’, the execution time will not increase. But it balance the total energy consumption values in time slots $T_5$ and $T_6$, and it allow better mapping.

The Figure 3 shows the example mapping process to choose alternate mapping to reduce the peak temperature for chip multiprocessor with two processors. Two hot tasks should not execute in two consecutive time slots on the same processor, and also no two hot tasks should execute at same time slot on both the processors. In the mapping section, we tried to do the same thing for chip multiprocessor with Row x Col processors arranged in grid fashion.

C. Energy Cap Enforced Modified Hu’s Approach (Cap-HU): schedule mix of hot tasks and cool tasks as much as possible in the current time slot with a cap on total energy consumption

Independent scheduling and mapping may not be able to reduce the temperature if the scheduler produce a schedule where many hot tasks are scheduled in the two consecutive time slots (or in the same time slot). So the scheduler should put a cap on number of hot task (or total energy consumption value) in a time slot. In this case, we try to enforce a cap (upper bound) on the total energy consumption values for all the

---

**Algorithm 1 Hu’s approach: Highest level first (HLF)**

**Input:** $N$ applications and $M$ processor.

1. $T_s = 1$ ($T_s$ is time slot)
2. while All tasks of all the applications are not scheduled do
3. Sort applications in non-increasing order of remaining unscheduled length.
4. if $M$ is greater than number of ready tasks then
5. Schedule ready task of all the application
6. else
7. Schedule ready tasks of the $M$ applications with highest remaining tasks of application
8. $T_s = T_s + 1$

---

**Fig. 2. Multiphase applications**

(a) With arbitrary execution time and arbitrary power consumption  
(b) Conversion of arbitrary execution time to unit execution time  
(c) With unit execution time and arbitrary energy consumption
V. MAPPING OF SCHEDULED TASKS TO CHIP MULTIPROCESSOR

In this section, we have described the mapping procedure of already scheduled tasks on to chip multiprocessor to reduce the peak temperature of the chip. Thus mapping output hugely depends on the considered architecture model. In our case, we have considered five mapping architecture models namely: random, greedy, row column, checker board (CB) and boundary fix checker board (BFCB). These models are described in subsections of this section. Algorithm 2 shows the high level mapping procedure of already scheduled task on to chip multiprocessor. This high level mapping procedure takes two inputs: (a) slot wise scheduled tasks for all the time slots and (b) architecture model of the $Row \times Col$ grid of processors. In step 1, it initialize time to zero, initialize map to null values and set temperature of the all the grid points to initial ambient temperature $T_{init}$. In main loop of the mapping procedure, for every time slot $T_s$, it map all the scheduled tasks of the slot $T_s$ to chip multiprocessors based on the specified architecture model. After the mapping process, it calculate the temperature for all the grid points (processors) of the chip multiprocessor by considering the current map, previous temperature of the grid points and ambient temperature using matrix based thermal prediction model as defined in equation 7 of Section III-B. In the main loop, it also calculate peak temperature for each time slots for the specified architecture model, and this peak temperature used for our result analysis.

Algorithm 3 shows common mapping procedure for all the scheduled tasks at time slot $T_s$ on to the chip multiprocessor based on the specified architectural model. It essentially sort all the tasks of the current time slot based on their energy consumption values, and also it sort all the locations of the grid based on the architecture model. If the model is greedy then all the location of the grid get sorted by temperature of grid location in the previous time slot. If the model is based on row col, checker board (CB) and boundary fix checker board (CFCB) then the all the locations of the grid get sorted based on the locations and the current values of toggle state. Toggle state is a binary variable, which value get toggled in every time slot. All the architectural models are described in the further subsections. Sorting base for the processor locations of different architecture models are given in code snippet except the random architecture model.

```c
bool BasedOnTemp(int A, int B) {
    return T_PR[A/R][A%C] < T_PR[B/R][B%C];
}
bool BasedOnRowCol(int A, int B) {
    return (A/R+Toggle)%2)!=(B/R+Toggle)%2;
}
bool BasedOnCB(int A, int B) {
    return ((A+Toggle)%2)!=(B+Toggle)%2;
}
bool BasedOnBFCB(int A, int B) {
    if(A/R==0||A/R==R-1||A%C==0||A%C==C-1) return 1;
    return ((A+Toggle)%2)!=(B+Toggle)%2;
}
```
Algorithm 3 Common mapping of scheduled tasks of slot $T_s$ on to chip multiprocessor based on the architecture model

**Inputs:** Model, $T$, $M$, $P_{pr}$, $[R,C]$, Scheduled Tasks at $T_s$

1: SortingBase[5]=(BasedOnRowCol, BasedOnTemp, BasedOnCB, BasedOnBFCS)
2: if Model!=Random then
3: Sort all the scheduled tasks at $T_s$ based on their energy consumption value
4: Sort all the processor locations based on Sorting based on SortingBase
5: Map all the sorted to task one by one to sorted location

---

**Fig. 6.** Peak temperature of the chip for different architecture mapping models at different time slots

**Fig. 7.** Peak temperature of Chip for different scheduling policies and mapping policies (Benchmark Mix 0)

**A. Random Model**

In this model, we map the scheduled tasks of the current time slot to a processor with out considering thermal issue and choose location for the scheduled task randomly. Every schedule tasks of the current time slot get mapped to randomly chosen processor location (with constraints of one task get mapped to one processor and vice verse in the same time slot).

**B. Greedy Model**

In this model, we map task with high energy consumption ($EC$) value to a processor with low predicted (or sensed) temperature in last time slot based on equation 7 in sorted order. This work based on assumption that: hot task mapped to cool location, so it will take time reach the temperature to a high level and cool task get mapped to hot place, there the task will not generate much heat to reach the temperature to a high level.

Suppose every processor have a temperature sensor and our mapper have access to this. So greedy model in general use this temperature value of processor instead of calculating the temperature. The greedy model is similar to dynamic thermal management (DTM) without avoiding mapping of task even if temperature exceed the dangerous value.

The greedy model is well suited for case where the effect of temperature of the self processor and effect of ambient temperature is significantly high as compared to effect of temperature of neighbors. This case happens when the feature size (manufacturing technology) is above 90nm.

**C. Row Column Model**

When the feature size (manufacturing technology) is below 28nm and effect of temperature of neighbors are significant as compared to self and ambient temperature, so greedy model is a bad choice. As in greedy model, it do not consider the temperature of the neighbor processors.

In row column model architecture, we do not consider the temperature of the grid points in the previous time slot, so we may not require to calculate (or predict) or sense the temperature of grid point to map the task. On the other hand, as a whole the grid points get logically divided in to alternative columns (rows) of hot processors and columns (rows) of cool processors. And in every time slot this logical hot processor columns (rows) and cool processor columns (rows) get exchanged. In this model, we hot task get mapped to cool processor columns (rows) of the current time slot. In the next time slot, position of hot columns and cool columns get exchanged. Figure 5(a), 5(d) and 5(g) shown position of hot processor column (shown in red) and cool processor column (shown in green) at time slot 1, 2 and 3 respectively.

**D. Checker Board Model (CB)**

In row column model, it only consider left or right neighbor in mapping. In this model, it consider four neighbors that is left, right, upper and down processors. The processor locations are arranged in a checker board like fashion of hot locations and cool locations as shown in Figure 5(b). Hot tasks get mapped to cool processor locations of the current time slot. In the next time slot, position of hot processor locations and cool processor locations get exchanged. Figure 5(b), 5(c) and 5(h) shown position of hot processors (shown in red) and cool processors (shown in green) at time slot 1, 2 and 3 respectively. Similar kind architecture model is used in Khdr et al. [21] to reduce peak temperature of the chip.

**E. Boundary Fix Checker Board Model (BFCS)**

As corner and boundary processors have less neighbors as compared to central processors, so we consider these processors as cool processors. In this model, the processor locations are arranged in a checker board like fashion except the boundary processors. Similar to CB model, in the next time slot the position of hot processor locations and cool processor...
locations get exchanged. Figure 5(c), 5(f) and 5(i) shown position of hot processor locations (shown in red) and cold processor locations (shown in green) at time slot 1, 2 and 3 respectively.

VI. EXPERIMENTAL EVALUATION AND ANALYSIS

We have used both synthetically generated data, and real benchmarks (like mf, swim, dct, ft, applu, equake, wupwise, jpegc, fma3d, apsi and heat) to evaluate our approaches. We have used Multi2sim simulator [5] to generate the phase wise power consumption behavior of application by setting 10k cycles as time slot (or quantum). We run the applications on top of Multi2sim simulator in a single core configuration (x86, 4 issues, out of order, 2 integer units, 2 floating point units) and calculated power consumption for different time slots using instruction level power model [27] and McPAT [28]. This slot wise power consumption data are used by our home grown simulator to evaluate scheduling and mapping approaches of multi phase applications on chip multiprocessor to reduce the peak temperature of the considered chip.

We have created two benchmark application instances mix Benchmark Mix 0 and Benchmark Mix 1. In each benchmark mix, we took 100 benchmarks and these are randomly from benchmark set. For Benchmark Mix 0, we have used benchmarks from set S1={mf, swim, dct, applu, equake, wupwise, apsi, dct, heat, matmul, covo1 and heat}, and for Benchmark Mix 1 benchmarks are from set S2={ mf, swim, dct, ft, applu, equake, jpegc, fma3d, apsi and heat}.

Figure 6 shows the peak temperature of the chip for different architecture models at different time slots for synthetic data (randomly generated application set of 100 applications and each application having upto 10 tasks and scheduled using HU’s approach. Result shows that when the neighborhood effect \( \text{neighboeff} = 0.0 \) (effect of matrix A of the equation 7 is negligible when chip feature size is large around 90nm technology), the greedy model perform better as compared to simple location based random, row col (RC), checker board (CB) and boundary fix checker board (BFBC) models. But the BFBC model also perform others when \( \text{neighboeff} = 0.35 \) and more. Interestingly the performance of greedy model is very bad as compared to random, RC and CB for higher values of \( \text{neighboeff} \). The cost of the greedy model is high as it needs to predict (or sense) the temperature of the grid in current time slot, and which is not a trivial.

Figure 7 shows time slot wise peak temperature of the considered multicore chip (with \( \text{neighboeff} = 0.35 \)) for four different scheduling (naive, TPS, HU and ModHU) approaches for Benchmark Mix 0. In all these cases, greedy mapping policy perform badly (in peak temperature reduction) as compared to random, row-col, checker board (CB) and boundary fix checker board (BFBC) mappings. Among all the five mappings, BFBC maintained low peak temperature in both HUs and ModHU’s scheduling even if \( C_{\text{setup}} \) is less in HU and ModHU as compared to naive and TPS scheduling approaches. In naive and TPS the peak temperature is high and also the system runs for extra amount of time as the scheduling is not optimal in term of execution time.

In the above example case, number of time slots required to execute the application instances of Benchmark Mix 0 are 794, 794, 985 and 1410 when HU, ModHU, naive and TPS scheduling used respectively. As we know non-preemptive scheduling of \( N \) applications with arbitrary execution time on \( M(>2) \) processors is not solvable in polynomial time.

So the TPS and naive use load balancing approach to schedule without considering the phase-wise behavior of applications. In our case, we take benefit of unit time phase wise behavior of applications and used critical path based scheduling (polynomal time algorithm in term of number of task \( n \), but not with number of application \( N \)) to produce optimal execution time. Even if both naive and TPS slot wise mapping of tasks to reduce the overall peak temperature by interleaving hot and cool applications, but performance in term of peak temperature reduction is not good when neighbor effect is high.

Figure 8, shows overall peak temperature of the chip for running Benchmark Mix 0 and Benchmark Mix 1, using all the five scheduling approaches and all the five mapping architecture models. We can see that BFBC is almost performing very good as compared to others. Performance of greedy approach is very bad in all the considered cases for both the benchmark mixes and all scheduling policies. The mapping model shows more than 40% less peak temperature for most of the cases. Others mapping policy random, row column and CB are almost performing similar to BFBC.

VII. CONCLUSION AND FUTURE WORK

Thermal hot spot and high temperature is an important issue in the current day deep sub micron chip multiprocessor. We have explored the benefit of different kind of temperature aware scheduling approaches and mappings of applications on to chip multiprocessor to reduce the peak temperature. As most of the application’s run time exhibit phase wise behavior, we have exploited the phase wise power consumption behavior of applications to schedule and map the applications to multicores to reduce the peak temperature. Critical path based scheduling in combination with simple location based mapping can reduce peak temperature of chip significantly without much increasing the execution time in executing phase wise applications on chip multiprocessor.

In our case, we have assumed model is an 90nm mesh architecture and the each processor are homogeneous and support on hardware threads. Supporting heterogeneous core and multiple hardware thread per processor may be a good extension to the work. Also including the effect of cache, memory hierarchy and network will be a great extension to the work. Integrated scheduling and mapping to reduce the peak temperature is really immediate extension of the same.

REFERENCES


1101