An oft-repeated benefit of multicore platforms over computationally-equivalent single-core platforms is increased energy efficiency and thermal dissipation. For these power benefits to be fully realized, a computer system must possess the ability to parallelize its computational workload across the multiple processing cores. However, parallel computation often comes at a cost of increasing the total, overall computation that the system must perform due to communication and synchronization overhead of the cooperating parallel processes. In this document, we explore the trade-off between parallelization on real-time applications and peak-system temperature minimization.

Very little research has addressed both parallelization and power-consumption issues [8], [2] and the work that exists in this area concentrates on energy consumption and has restriction on the real-time task model [8]. While minimizing the energy consumption and peak temperature of a system are related problems, the solutions for addressing these problems are fundamentally different. For example, it is well known that in a system with dynamic voltage and frequency scaling capabilities (DVFS) the optimal frequency assignment of each processing core (in the presence of sequential real-time tasks) for the purpose of energy minimization is quite different than the frequency assignment for peak-temperature minimization (e.g., see Wang and Chen [12]). Clearly, since sequential tasks are a special, restrictive case of parallel tasks, this example continues to exhibit the differences in the objective for the parallel real-time task model. Unfortunately, while numerous research papers have addressed peak temperature minimization in systems comprised of non-parallelizable real-time jobs (see [11] for a survey), to the best of our knowledge, no prior research exists concerning job parallelization and peak temperature in the real-time literature. We report in this document open problems related to the scheduling of parallel real-time tasks upon multiprocessors constrained the peak temperature. We will first introduce, in sections II-A & II-B, important concepts for thermal and parallel systems, respectively.

I. INTRODUCTION

A. Thermal Model

We consider a multicore processing platform with a set of heat sinks for heat dissipation. Using the notation of Fisher et al. [5] in the following, the multicore processing platform $\mathcal{M}$ consists of $M$ cores labeled by the index number, $\mathcal{M} = \{1, 2, 3, \ldots, M\}$. The thermal conductance between Cores $j$ and $\ell$ in $\mathcal{M}$ is $G_{j,\ell}$, where $G_{j,\ell} = G_{\ell,j}$. We assume that the capacitance of Core $j$ in $\mathcal{M}$ is $C_j$. Suppose that the thermal conductance of a core dissipating heat to the environment is $G^\gamma$. For simplicity and brevity, we do not include heat sinks in our description; however, they can easily be incorporated into the model.

The temperature of Core $j$ is defined as $\Theta_j(t)$ and $\Theta_a(t)$. For the current problem formulation, we assume that the ambient temperature $\Theta_a$ is fixed. We also define $\Psi_j(t)$ as the power consumption on Core $j$ at time $t$. As a starting point, we assume that $\Psi_j(t)$ equals $\alpha s_j^\gamma$, where $s_j$ is the execution speed of Core $j$ and both $\gamma$ ($\leq 3$) and $\alpha$ are processor-dependent constants. In this initial problem description, we focus only on dynamic-power consumption and ignore leakage power; however, to be realistic, a complete solution would consider both dynamic and static power. We hope that solutions for our posed open question involving dynamic power can be extended to the static-power setting.

Informally, the rate of change in the temperature on a core is proportional to the power consumption times the quantity of the heating coefficient minus the cooling coefficients times the quantity of the temperature gradients among the core and its neighboring cores. The heating/cooling process may be calculated by using the duality principle between electrical and thermal circuits and standard theory of electrical circuits:

$$C_j \frac{d\Theta_j(t)}{dt} = \Psi_j(t) - \sum_{\ell \in \mathcal{M}} G_{j,\ell}(\Theta_j(t) - \Theta_\ell(t)) - C^\gamma(\Theta_j(t) - \Theta_a) \tag{1a}$$

where $\frac{d\Theta_j(t)}{dt}$ is the derivative of the temperature on Core $j$. 

II. MODELS
B. Parallel Task Models

We deal with jobs which may be executed on different processors at the very same instant, in which case we say that job parallelism is allowed. Various kind of task model exist, Goossens et al. [6] adapted parallel terminology [1] to recurrent (real-time) tasks as follows.

Definition 1 (Rigid, Moldable and Malleable Job): A job is said to be (i) rigid if the number of processors assigned to this job is specified externally to the scheduler a priori, and does not change throughout its execution; (ii) moldable if the number of processors assigned to this job is determined by the scheduler, and does not change throughout its execution; (iii) malleable if the number of processors assigned to this job can be changed by the scheduler during the job’s execution.

At task level the literature distinguish between at least two kinds of parallelism.

- **Multithread.** Each task is sequence of phases, each phase is composed of several threads, each thread requires a single processor for execution and can be scheduled simultaneously [10]. A particular case is the Fork-Join task model where task begins as a single master thread that executes sequentially until it encounters the first fork construct, where it splits into multiple parallel threads which execute the parallelizable part of the computation [9] and so on.

- **Gang.** Each task corresponds to \( e \times k \) rectangle where \( e \) is the execution time requirement and \( k \) the number of required processors with the restriction the \( k \) processors execute task in unison [7].

Assuming that a job \( J_t \) has a processing requirement of \( e_t \) and is assigned to \( k_t \) processors for parallel execution, then several model are proposed in the literature to characterize the multiprocessor speedup vector \( \Gamma_t = (\gamma_{t,1}, \ldots, \gamma_{t,m}) \) with the interpretation that job \( J_t \) that executes for \( t \) units on \( j \) processors completes \( \gamma_{t,j} \cdot t \) units of execution.

- **sub linear speedup ratio** [8] requires that \( 1 < \frac{\gamma_{j,j'}}{\gamma_{j,1}} < \frac{j'}{j} \) where \( j < j' \).

- **work-limited parallelism** [3] \( \gamma'_{i,j} > \frac{\gamma_{i,j}}{\gamma_{i,j'}} \) and \( \gamma_{i,j'} = \gamma_{i,j} \cdot j' - j \). Each task is sequence of phases, each phase executes for \( \ell \) time units on \( j \) processors completes \( \gamma_{t,j} \cdot \ell \) units of execution.

- **communication model** [4] requires that \( \gamma_{t,j} = \frac{e_t}{C + (\ell-1)C} \) where \( C \) is the constant communication overhead cost.

III. OPEN PROBLEMS

Informally, our main open problem is:

For each Core \( j \in \mathcal{M} \), determine the speed/frequency assignment \( s_j \) that minimizes the peak system temperature (i.e., minimize \( \max_{\ell \geq 0} \Theta_{\ell}(1) \)) such that all real-time parallel jobs in a recurrent task system meet their deadline.

Furthermore, we pose the above problem for each of the parallel job models (i.e., rigid/moldable/malleable).

To solve the above general problem, we must answer the following subproblems:

**SP1 Development of Parallel Job Schedulability Analysis for Uniform Multiprocessor Platforms.** Since each core executes at a potentially different speed and any job may execute on any core, the uniform heterogeneous multiprocessor platform model is an appropriate processing abstraction. However, to the best of our knowledge no schedulability analysis exists for the parallel real-time job setting.

**SP2 Development of Thermal-Aware Online Real-Time Scheduling Algorithms.** The locality of the cores and their thermal properties are significant factors in minimizing peak temperature. Thus, a single parallel job may generate heat from multiple sources which may have some complex interaction. Therefore, an example showing the benefit of parallelism for peak-temperature minimization would be interesting and we are interested in online algorithms for determining how a parallel job should be “spread” across processors to optimize our objective.

**SP3 Development of Thermal-Aware Frequency Assignment Schemes.** An offline allocation algorithm or online scheduling algorithm is necessary to decide on what are the values of \( s_j(t) \) for each \( j \in \mathcal{M} \). In the offline setting, the frequency/speed allocation algorithm should determine the minimum assignment that respects deadlines and optimizes the objective function. In the online setting, the core speed could dynamically change.

REFERENCES