Thermal, Power, and Co-location Aware Resource Allocation in Heterogeneous High Performance Computing Systems

Mark A. Oxley*, Eric Jonardi*, Sudeep Pasricha*†, Anthony A. Maciejewski*,
Gregory A. Koenig‡, and Howard Jay Siegel*†

*Department of Electrical and Computer Engineering
†Department of Computer Science
Colorado State University, Fort Collins, Colorado 80523, USA

‡Oak Ridge National Laboratory
Oak Ridge, Tennessee 37830, USA
{mark.oxley,eric.jonardi,sudeep,aam,hj}@colostate.edu, koenig@ornl.gov

***Abstract*—The rapid increase in power consumption of high performance computing (HPC) systems has led to an increase in the amount of cooling resources required to operate these facilities at a reliable threshold. The cooling systems contribute a large portion of the total power consumption of the facility, thus driving up the costs of providing power to these facilities. In addition, when cores sharing resources (e.g., last-level cache) execute applications at the same time, they can experience contention and therefore performance degradation. By taking a holistic approach to HPC facility management through intelligently allocating both computing and cooling resources, the performance of the HPC system can be maximized by considering co-location while obeying power consumption and thermal constraints. The performance of the system is quantified as the total reward earned from completing tasks by their individual deadlines. We propose three novel resource allocation techniques to maximize performance under power and thermal constraints when considering co-location effects: (1) a greedy heuristic, (2) a genetic algorithm technique used in combination with a new local search technique that guarantees the power and thermal constraints, and (3) a non-linear programming based approach (from previous work), adapted to consider co-location effects.**

***Keywords*-heterogeneous computing; resource management; thermal-aware computing; power-aware computing; data center; DVFS; memory interference**

I. Introduction

The power consumption of high-performance computing (HPC) systems and data centers is increasing rapidly, leading to an increase in the amount of cooling resources required to operate these HPC systems at a safe temperature threshold. The increase of power consumption to operate today's facilities has led to the design of power-efficient systems such as the TSUBAME-KFC at the Tokyo Institute of Technology that tops the Green500 list with an efficiency of 4.5 GFLOPS/Watt [1]. Extrapolating the power consumption of the TSUBAME-KFC system to exascale, it would consume 222 MW, which equates to approximately $145 million per year in electricity costs in the United States. The Defense Advanced Research Projects Agency (DARPA) has set the target for an exaflop system to consume no more than 20 MW [2], an order of magnitude of power consumption lower than what can be achieved using today's computing infrastructure.

One technique that can be used to reduce the power consumption and also manage thermal issues in large-scale computing systems, such as data centers, is power-aware and thermal-aware resource allocation. This approach involves exploiting system characteristics such as the heterogeneity among different servers (offering different levels of performance and power consumption), dynamic voltage and frequency scaling (DVFS) in cores, and the interactions between temperatures of the compute nodes and computer room air conditioning (CRAC) units. By distributing workloads, configuring DVFS, and setting CRAC thermostats in an intelligent manner, it becomes possible to efficiently balance the compute performance with the power consumption and temperature profiles of the HPC facility.

By the law of conservation of energy, the power consumed by servers is dissipated as heat. This heat must be removed by the cooling infrastructure so that compute nodes can safely operate beneath their specified redline temperatures (the maximum safe operating temperatures). The more power that is consumed by compute nodes, the more power CRAC units must consume to remove additional heat and maintain the redline temperatures. Cores within compute nodes are DVFS-enabled, with performance states (P-states) that provide a trade-off between compute performance and power consumed by each core. By intelligently configuring P-states of cores, power consumption can be reduced at the compute nodes, which in turn results in less power that is needed by the CRAC units to cool the facility.

As the number of cores increase in emerging multicore processors, access contention in shared memory (e.g., last-level caches, DRAM) can have a pronounced impact on the

978-1-4799-6177-1/14/$31.00

execution time of workloads [3]. We model this interference as a performance degradation in the execution rate of tasks when different cores in the same multicore processor are executing tasks at the same time. The performance degradation can be severe when memory-intensive tasks running on separate cores of the same processor access the shared memory simultaneously. Thus, it is intuitive that to minimize interference, one should co-locate memory-intensive tasks with compute-intensive tasks (that have relatively fewer memory accesses), if possible.

We consider a steady-state model of a data center, where task flow rates, temperatures at compute nodes and CRAC units, and the power consumption of the computing system and CRAC units remain invariant. The performance of the data center is measured by the reward collected from completing tasks by their individual deadlines, where reward represents the worth of completing that task to the system. In the steady-state this is equivalent to the *reward rate* collected. Our resource management techniques mitigate the impact of co-location interference by maximizing *co-located reward rate*, a measure for estimating reward rate when considering co-location interference. By taking a holistic approach to the control of such a facility, we maximize the co-located reward rate earned while ensuring that the compute nodes do not exceed their redline temperatures and the total power consumed by the compute nodes and CRAC units do not exceed a given constraint. To solve this optimization problem, we design a new greedy heuristic, a novel genetic algorithm (GA) with local search, and improve a non-linear programming (NLP) approach from our previous work [4] to consider the effect of co-locating tasks on multicore processors.

In summary, we make the following novel contributions:

- Derivation of a new detailed model of a heterogeneous HPC system that considers the power consumption of both the compute servers and cooling system, thermal constraints, DVFS P-states, memory-intensity of tasks, and co-location interference.
- Design of a greedy heuristic, a genetic algorithm with local search, and an adaptation of a previously proposed non-linear programming approach that is the first work to our knowledge to consider co-location while obeying power consumption and thermal constraints.
- Analysis and comparison of our resource allocation techniques with prior work that does not consider co-location when optimizing for a power and thermal constrained data center.
- Analysis of the performance of our proposed techniques on three different platform sizes.

The rest of the paper is organized as follows. We discuss related work in Section II. In Section III, we explain our models for compute nodes, CRAC units, and workload as well as how we consider co-location interference. Section IV describes our proposed resource allocation techniques. Our evaluation setup and results are in Sections V and VI. In Section VII, we conclude and discuss ideas for future work.

II. Related Work

The power consumption of a data center's cooling system can consume approximately 50% of the total power consumed by the facility [5]. Proper thermal management of a data center can therefore result in large savings in dollar expenditures due to reductions in energy consumption of the compute and cooling infrastructure. Hot spots in the data center can be mitigated in several ways, such as intelligent workload distribution, throttling of compute servers (typically using DVFS), and better management of CRAC units. The techniques proposed in [6] distribute the workload across a data center to minimize inefficiencies in heat removal and hot spot formation, so as to minimize cooling costs. Throttling the performance of servers (or shutting them down) is another technique for controlling the temperatures in an HPC facility. The research in [7] analyzes whether it is more energy efficient to distribute the workload evenly to avoid hot spots, or to concentrate the workload on a small number of servers to be able to deactivate many servers. Methods for controlling the on/off state of CRAC units to minimize hot spots in a data center and save cooling energy by reducing air flow in overprovisioned areas are proposed in [8]. These works perform thermal-aware resource allocation by examining only one of either workload distribution, throttling servers, or managing CRAC units, whereas our work simultaneously considers workload distribution, throttling through DVFS, and CRAC thermostat control.

The thermal-aware approach designed in [9] distributes the workload to servers in a spatial manner such that hot spots are minimized, allowing the CRAC thermostats to be set to higher temperatures so that cooling power is reduced. That work does not consider throttling of servers to reduce the power consumption (and heat generation) of compute nodes. Our previous work considers managing the workload distribution, throttling of servers, and managing CRAC units simultaneously. The goal of the study in [4] is to maximize the reward rate earned in a data center under power consumption and thermal constraints. A novel non-linear programming technique is designed to solve the resource allocation problem. We build upon this work by enhancing the resource allocation technique to consider co-location interference, and by designing new resource allocation techniques that scale better for larger system sizes.

In situations where tasks are co-located on the various cores of an individual processor, the primary sources of performance degradation are the contention for shared caches and memory bandwidth. Depending on the workload characteristics of the co-located task, the performance degradation due to co-location can range from negligible to

severe [3]. Prediction of the impact of co-location requires an understanding of the memory system requirements of each task and how each task would affect and be affected by the other tasks with which it is co-located. This understanding can be obtained by either characterizing individual tasks using synthetic testing structures [10], or experimentally by pairing tasks together [3]. Co-location predictions could be used to reserve memory bandwidth, such as the MemGuard technique described in [11] that predicts the memory interference of an application and uses that knowledge in a scheduler to avoid co-location effects. Our research uses the knowledge and experimental data presented in those works to consider the co-location effects at a larger scale than the single-processor analyses in those works.

III. System Model

A. Overview

Our model of a data center and workload builds on the model proposed in [4]. We assume the data center is configured in a hot aisle/cold aisle fashion (Fig. 1). In such a configuration, cold air is supplied from the CRAC units to a cold aisle through perforated floor tiles that face the inlets of the compute nodes. The compute nodes consume power and expel hot air through the opposite end to a hot aisle. The CRAC units draw the hot air from the hot aisles to cool. Our model can support isolated and non-isolated aisle configurations.

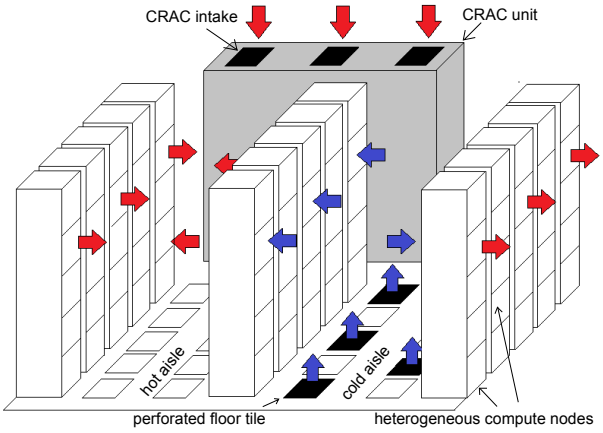

Figure 1. Data center in hot aisle/cold aisle configuration.

B. Compute Nodes

The number of compute nodes in the data center is NN, and each compute node j belongs to a compute node type $NT(j)$. We assume a heterogeneous data center with the number of compute node types equal to NNT, and compute nodes that belong to a specific compute node type $NT(j)$ are identical and contain the same number of cores, power characteristics, and performance characteristics. Cores within a compute node are homogeneous, and we assume the cores can run in individual P-states that provide a tradeoff between power consumption and performance [12]. The total number of cores in the data center is NC, and $CT(k)$ is the type of the compute node to which core k belongs.

C. Workload

We assume that we have a set of T known task types. The arrival rate of tasks of type i is given by λ_i. A reward r_i is obtained for completing a task of type i by its individual deadline d_i, relative to its arrival time.

As our system is heterogeneous, the power and performance characteristics of the compute node types are different. Therefore, tasks of different types can have different execution rates on different node types and P-states. We assume that we know the estimated computational speed (ECS) of any task of type i on a core of type j in P-state k, ECS(i,j,k) (i.e., number of tasks per second). In an actual system, these can be approximated using historical, experimental, or analytical techniques [13, 14, 15]; in Section V we discuss the values used for our evaluation.

Our goal is to assign a *desired fraction* of time each core k will spend executing tasks of type i, denoted $DF(i,k)$, and the P-state each core k is configured to when executing tasks of type i, denoted $PS(i,k)$, to maximize the reward we can obtain and meet the power and thermal constraints of the system. Given $DF(i,k)$ and $PS(i,k)$, we calculate the *execution rate* of tasks of type i on core k, $ER(i,k)$, as

$$ER(i,k) = DF(i,k) \cdot ECS(i, CT(k), PS(i,k)). \quad (1)$$

The total *reward rate* earned, denoted RR, without considering co-location interference is

$$RR = \sum_{i=1}^{T} \left(r_i \cdot min(\sum_{k=1}^{NC} ER(i,k), \lambda_i) \right). \quad (2)$$

Equation (2) states that the reward rate for a given task type i is the product of the reward earned and its execution rate. The min function in Equation (2) enforces the intuitive constraint that if a task is assigned for execution at a faster rate than its arrival rate λ_i, additional reward is not earned.

Reward is earned only when tasks are completed by their deadline. We have no way of guaranteeing deadlines are met in our steady-state model, but we can help minimize the number of deadlines that would be missed by eliminating any task type/core type/P-state combinations from consideration that would result in a missed deadline even if a task of type i starts immediately after its arrival, i.e., we eliminate combinations from consideration that violate

$$\frac{1}{ECS(i, CT(k), PS(i,k)} \geq d_i. \quad (3)$$

Another constraint that must be enforced is to ensure cores do not spend greater than 100% of the time executing tasks. If a core k is spending greater than 100% of the time executing tasks, we normalize the $DF(i,k)$ values on that core to guarantee this constraint. Normalization is performed

by summing the desired fractions of time all task types spend executing on a given core k, denoted $SumDF(k)$, and dividing the desired fraction of time values for all T task types by $SumDF(k)$, forcing their sum to equal 100%. This normalization procedure is performed before calculating execution rates (Equations 1, 9, and 10) and the power consumption of a node (Equation 4, Section III-D). Normalization is not performed on a core if the value of $SumDF(k)$ is less than 100%.

D. Power and Thermal Model

We consider the overhead power consumption of a compute node (e.g., from main memory, disks, fans, NICs) in addition to the power consumption of the CPU cores. We assume that CPU cores are able to change P-states over time depending on what task type is currently being executed, and that the time associated with switching P-states is negligible in comparison to the execution time of tasks. The power consumed by cores is a function of the task type being executed in addition to the P-state in which the core is executing a task. Let $O(j)$ be the overhead power consumption of compute node j, let APC(i,$NT(j)$,$PS(i,k)$) be the average power consumed by cores in a node of type $NT(j)$ executing tasks of type i in P-state $PS(i,k)$, and $NCN(j)$ be the set of cores in node j. In the steady-state, we calculate the power consumption of node j, $PN(j)$, as

$$PN(j) = O(j) + \sum_{k \in NCN(j)} \sum_{i=1}^{T} APC(i, NT(j), PS(i,k)) \cdot DF(i,k). \quad (4)$$

The power consumed at a CRAC unit is a function of the heat removed at that CRAC unit in addition to the Coefficient of Performance (CoP) [16] of the CRAC unit. Let NCR be the total number of CRAC units in the data center, $TC^{in}(i)$ be the inlet temperature of CRAC unit i, $TC^{out}(i)$ be the outlet temperature of CRAC unit i, ρ be the density of air, C be the specific heat capacity of air, and $AFC(i)$ be the air flow rate of CRAC unit i. The power consumed by CRAC unit i, $PC(i)$, is calculated as [16]

$$PC(i) = \frac{\rho \cdot C \cdot AFC(i) \cdot (TC^{in}(i) - TC^{out}(i))}{CoP(TC^{out}(i))}. \quad (5)$$

To calculate the steady-state temperatures at compute nodes and CRAC units, we use the Abstract Heat Flow Model from [17]. Let $TN^{in}(i)$ be the inlet temperature at compute node i and $TN^{out}(i)$ be the outlet temperature at compute node i. The outlet temperature at compute node i is a function of the inlet temperature, the power consumed, and the air flow rate of the node $AFN(i)$, calculated as

$$TN^{out}(i) = TN^{in}(i) + PN(j) / (\rho \cdot C \cdot AFN(i)). \quad (6)$$

Let $\mathbf{T^{out}}$ and $\mathbf{T^{in}}$ be the vectors of outlet and inlet temperatures of CRAC units and compute nodes. Also, let $\mathbf{A}$ be a matrix of cross-interference coefficients where each element $\alpha(i,j)$ represents the percentage of heat transferred from CRAC unit or node i to CRAC unit or node j. We can then calculate our temperatures as [17]

$$\mathbf{T^{in}} = \mathbf{A}\mathbf{T^{out}}. \quad (7)$$

If we let $\mathbf{T^{redline}}$ be the vector of redline temperatures, the thermal constraint is the element-wise inequality

$$\mathbf{T^{in}} \leq \mathbf{T^{redline}}. \quad (8)$$

Table I
% PERFORMANCE DEGRADATION DUE TO TASK CO-LOCATION

memory intensity of task type	heavy	medium	light
heavy	100	30	5
medium	50	20	3
light	6	4	1

E. Co-Location Interference

Tasks competing for shared memory in multicore processors can cause severe performance degradation, especially when competing tasks are memory-intensive [3]. The memory intensity of a task refers to the ratio of operations performed in the CPU (e.g., floating point) to the number of bytes accessed in main memory. Based on experimental data from [3] that measures the execution time slowdown of some SPEC benchmarks, we create the degradation table shown in Table I that gives the performance degradation for a task type of a categorized memory intensity in row i when co-located with another task type of a given memory intensity in column j, given by $PD(i,j)$. For simplicity, we assume that a task type's memory-intensity is represented by one of three categories: heavy, medium, or light. For example, when a "heavy" task type is co-located with a "medium" task type, the PD value of 30% (from Table I) corresponds to a 30% increase in execution time.

We can probabilistically determine if a task type i would be executing on a core k by considering the $DF(i,k)$ value as the probability that core k is running a given task type i. Using these values, we can calculate the probability that cores within the same processor are executing any combination of task types to help estimate performance degradation caused by co-location. On a dual-core processor with cores k and l, we can estimate the co-located execution time of task type i on core k, $CET^{dual}(i,k)$, as

$$CET^{dual}(i,k) = \frac{1}{ER(i,k)} \left(1 + \sum_{t=1}^{T} (DF(t,l) \cdot PD(i,t)) \right). \quad (9)$$

Because execution rate is the reciprocal of execution time, the co-located execution rate for a dual-core processor, $CER^{dual}(i,k)$, is $1/CET^{dual}(i,k)$. The summation in Equation 9 represents the the slowdown observed of task i on core k when the co-located core l is executing task types assigned to it, weighted by the desired fractions of

time those tasks would execute on core l.

With a processor containing more than two cores, the performance degradation is additive based on the combination of task types executing on the cores co-located with core k. For example, let us assume a processor with three cores k, x, and y and a "heavy" task type is executing on core k. If assigning a "heavy" task type to core x increases the execution time of the task type on core k by 100% (from Table I), then assigning a "medium" task to core y increases the original execution time of the task type on core k by an additional 30% (from Table I), resulting in a total execution time increase of 130%. We observed this trend in experimental data from our recent work (e.g., the Streamcluster application from the PARSEC benchmark suite) [18]. Thus, the co-located execution time of task type i on core k on a quad-core processor with other cores x, y, and z, $CET^{quad}(i,k)$, is

$$CET^{quad}(i,k) = \frac{1}{ER(i,k)} \cdot \left(1 + \sum_{a=1}^{T}\sum_{b=1}^{T}\sum_{c=1}^{T}(DF(a,x)\cdot DF(b,y)\cdot DF(c,z)\cdot[PD(i,a)+PD(i,b)+PD(i,c)])\right) \tag{10}$$

The summation in Equation 10 represents the probability that task type a is executing on core x, task type b is executing on core y, and task type c is executing on core z multiplied by the sum of the performance degradation values of those particular task types. Again, execution rate is the reciprocal of execution time, therefore the co-located execution rate for a quad-core processor, $CER^{quad}(i,k)$, is $1/CET^{quad}(i,k)$.

Let $CER(i,k)$ be equal to $CER^{dual}(i,k)$ if k is on a dual-core processor and equal to $CER^{quad}(i,k)$ if k is on a quad-core processor. To assist our resource management techniques in estimating actual reward rate under the effects of co-location interference, we introduce a new objective called *co-located reward rate* (CRR), calculated as

$$CRR = \sum_{i=1}^{T}\left(r_i \cdot min(\sum_{k=1}^{NC} CER(i,k), \lambda_i)\right). \tag{11}$$

Thus, $CRR \leq RR$.

The goal of this study is to maximize CRR when subject to power consumption and thermal constraints. The total power consumed by both CRAC units and compute nodes must be less than the power constraint, denoted ϕ. The thermal constraint is defined in (8). In the next section, we describe several approaches to solve this problem.

IV. Heuristics

A. Greedy Heuristic

We designed a greedy approach similar to those in [19, 20] to assign task types to cores. Our greedy heuristic (Alg. 1) iteratively assigns task types to cores to find the most efficient mapping, where we define efficiency for allocating a particular task type to a core as the ratio of performance (ECS) to the power consumption. For a task of type i on a node of type j in P-state k, we define the efficiency of that mapping, EFF(i,j,k), as

$$EFF(i,j,k) = ECS(i,j,k)/APC(i,j,k). \tag{12}$$

Algorithm 1 Pseudo-code for our greedy heuristic

1. find most efficient P-state for all task type/node type pairs
2. sort all task type/node type pairs by efficiency
3. **while** power constraint not violated **do**
4. choose first task type/node type pair
5. assign 100% desired fraction of time for selected task type to a single core from selected node type
6. remove core from future consideration
7. **if** no cores within selected node type available
8. remove task type/node type pair from consideration
9. set CRAC outlet temperatures to hottest temperatures such that thermal constraints are met
10. **end while**

We start by finding the P-states with the highest EFF values for all task types and node types (line 1); all other P-states are not considered. All $T \times NNT$ task type to node type pairings are then sorted by their efficiency in descending order (line 2). At each iteration of the heuristic, the first pairing (i.e., most efficient) is selected and a single core of the chosen node type is assigned the designated task type by assigning that core a 100% desired fraction of time for that task type (lines 4-5). After making an assignment, that core is removed from consideration (line 6). If there are no cores available within the chosen node type (i.e., all cores within nodes of that node type been assigned a task type), that pairing is removed from consideration (lines 7-8). The CRAC outlet temperatures are set to the redline temperature, and then the outlet temperatures of all CRAC units are iteratively decreased by one degree until the thermal constraints are met (line 9). The algorithm repeats using the next pairing until the power constraint is violated or the execution rates for all task types meet or exceed their arrival rates.

B. Genetic Algorithm

1) Overview: We also designed a genetic algorithm (GA) based on the Genitor GA [19, 21] to solve our optimization problem. Our genetic algorithm in this study operates on a population of 200 chromosomes. Each chromosome represents one possible solution (i.e., a complete resource allocation). A chromosome consists of a collection of $|NC \times T|$ genes, where each gene is a pair of $DF(i,k)$ and $PS(i,k)$ values representing the desired fraction of time and P-state of a task type/core combination. The initial population is

Algorithm 2 Pseudo-code for our local search technique

```
1. while power and thermal constraints not met do
2.     set CRAC outlet temperatures to hottest temperatures
       such that thermal constraints are met
3.     find node with highest temperature (node j)
4.     while node j is hottest node do
5.         choose random core/task type
           combination from node j
6.         increase P-state assignment by 1
           if not already in maximum P-state
7.     end while
8. end while
```

generated by assigning random desired fractions of time and P-states to each gene within the chromosome, and then normalizing the desired fractions of time so that cores cannot spend greater than 100% of their time executing tasks. After the initial population generation, the chromosomes in the population are evaluated and ranked by co-located reward rate (CRR). We then perform crossover and mutation that alter existing solutions to generate offspring chromosomes. After the offspring are generated, local search is performed on the offspring to meet the power consumption and thermal constraints. The population is then evaluated and ranked, and subsequently the population is trimmed to its original size by eliminating the least-fit chromosomes.

2) Crossover and Mutation: Crossover starts by selecting two parent chromosomes using a linear bias [21] and two points are generated such that $x < y \leq |NC|$. All genes for cores ranging from x to y are swapped among parent chromosomes to create two offspring solutions.

Mutation is probabilistically performed on offspring chromosomes to introduce perturbations in the genes, allowing a broader search. Offspring chromosomes have a probability p_m of being mutated (empirically set to 0.1). If a chromosome is selected for mutation, each gene has a probability p_g of being mutated (empirically set to 0.05). If a gene is selected for mutation, the desired fraction of time and P-state for its task type on this core are set to random values. Then the chromosome is normalized so that the desired fractions of time for all cores sum to 100%.

3) Local Search: We perform a local search on offspring chromosomes that sets CRAC outlet temperatures and P-states such that the power consumption and thermal constraints are met. The pseudo-code for our local search is given in Alg. 2. The step described in line 2 is performed by setting the CRAC outlet temperatures to the redline temperature, and then the outlet temperatures of all CRAC units are iteratively decreased by one degree until the thermal constraints are met. The steps performed in lines 3 to 6 have a two-fold effect to reduce power consumption. First, increasing the P-state assignment (i.e., reducing the power and frequency) directly reduces the power consumption of the node. Second, the CRAC units may be able to run with a higher outlet temperature, reducing the overall power required to maintain redline temperatures.

C. Non-linear Programming Approach

We adapt a power and thermal-aware approach from [4] and improve it to include the effects of co-location by maximizing co-located reward rate instead of just maximizing reward rate as done in [4]. The problem in [4] is formulated as a non-linear program and then solved using approximations, heuristics, and linear programming. We refer the reader to [4] for the full details of this technique, but give a brief summary below.

The NLP technique is divided into three steps to solve the formulated mixed integer non-linear program for maximizing reward rate (Equation (2)) while obeying the power and thermal constraints. The first step relaxes the integer constraint on P-states (by assuming continuous P-states) and solves for the CRAC outlet temperatures and power consumption of compute cores such that reward rate is maximized and the power and thermal constraints are met. Using the core power consumption values obtained from the first step, the second step uses a simple heuristic to round the continuous power consumption values of cores to discrete P-states, while maintaining the power and thermal constraints. Lastly, a linear program is solved to maximize reward rate assuming the CRAC outlet temperatures from the first step and the discrete P-state assignments from the second step.

We improve upon this approach in two ways. First, we incorporate the knowledge of the memory intensity of tasks by considering the APC matrix (i.e., a core consumes a different amount of power based on task type) instead of assuming all task types consume the same amount of power in a given P-state. Also, the scaling of ECS values for different P-states is now a function of memory intensity in addition to clock frequency. Second, we incorporate knowledge of co-location interference in this algorithm by modifying the objective function to maximize co-located reward rate instead of reward rate in the first and third steps. We show in Section VI how the modified algorithm (NLPCL) makes significantly different resource allocation decisions than NLP because it considers co-location interference.

V. Evaluation Setup

A. Overview

In our simulations, we consider three heterogeneous platforms of different sizes. The *small* platform consists of three CRAC units, eight task types, and 150 nodes where each node is uniformly selected from two node types. The *medium* platform uses four CRAC units, 500 nodes (belonging to one of of three node types) and ten task types. The *large* platform uses 60 CRAC units, 5,000 nodes (belonging to one of four node types), and sixteen task types. The power characteristics of our different node types

Table II
NODE TYPES USED IN SIMULATIONS

node type	Fujitsu TX140	IBM x3200	Fujitsu RX100	Acer AT110
overhead power (W)	18.9	75.2	35	21.3
number of cores	4	2	4	4
number of P-states	14	4	11	16
power consumption of a core in P-state 0 (W)	9.1	20.9	22.4	18.55

were obtained from SPECpower_ssj2008 results [22] (see Table II), and the workload characteristics were obtained from SPEC_CPU2006 [23] results for the node types listed in Table II. The results for these machines using those benchmarks were collected between the years 2010 and 2012. We set the power constraint (ϕ) equal to 60% of the maximum power of the system, i.e., when all cores are executing the most CPU-intensive task type in P-state 0 with the CRAC units set to highest temperatures such that all nodes maintain redline temperatures. Simulations were implemented in a custom C++ environment on a laptop computer with a Core i7-4700HQ CPU and 8 GB of RAM in a VMware virtual machine running Ubuntu 12.04.

B. Workload

The task types and corresponding ECS matrix entries for each node type in P-state 0 are obtained from SPEC_CPU2006 results [23]. We use execution rate results for *lbm*, *bwaves*, *sphinx3*, *milc*,*cactusADM*, *calculix*, *wrf*, and *tonto* for the task types on the small platform size, add *povray* and *dealII* for the medium platform size, and then add *zeusmp*, *gromacs*, *leslie3d*, *namd*, *soplex*, and *GemsFDTD* for the large platform size. These benchmarks were chosen to enable a diverse portfolio of consumer and scientific applications on which to evaluate our optimization frameworks. Experimental data from [11] is used to quantify the memory intensity of these benchmarks. We scale the ECS values for a task type in other P-states based on the clock frequencies of the P-states and the memory intensity of the task type. Based on experimental data from [24], the performance of the most memory-intensive SPEC_CPU2006 application (*lbm*) scaled approximately 30% to that of the clock frequency. For example, halving the frequency resulted in only a 30% increase in execution time. The most CPU-intensive application (*povray*) scaled at approximately 100% to that of the clock frequency. We scale the ECS values for task types in P-states based on the clock frequency and memory-intensity values from [11].

The reward for task types are generated based on a uniform random variable in the range [0.5,1]. The d_i values, arrival rate λ_i values, and thermal coefficient matrix $\mathbf{A}$ are generated using the same techniques as in [4].

C. CRAC Units

In this study, we assume that the CRAC units are homogeneous. The Coefficient of Performance (CoP) for a CRAC unit is a function of its outlet temperature, τ, given by $CoP(\tau) = 0.0068\tau^2 + 0.0008\tau + 0.458$ [16] . The air flow rate of each CRAC unit is set so that the sum of the air flow rates of the compute nodes is equal to the sum of the flow rates of the CRAC units [4].

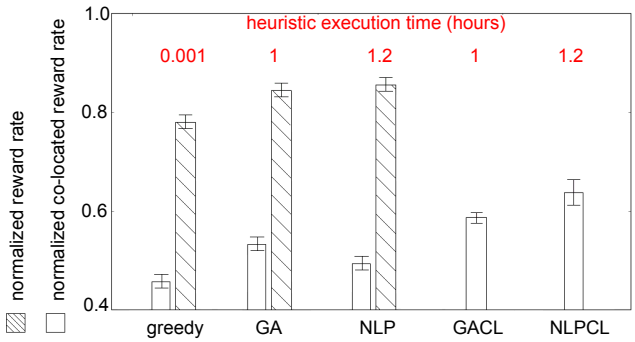

Figure 2. Comparison of co-located reward rate earned (non-hashed bars) by greedy, GA, and NLP techniques on the small platform size, and comparison of reward rate earned (hashed bars) by all techniques on the small platform size

VI. RESULTS

For our first experiment, we examine the value of accounting for co-location in our genetic algorithm and non-linear programming techniques. We denote *GA* as our genetic algorithm that uses reward rate to evaluate chromosomes, and *GACL* as a variation that uses co-located reward rate to evaluate chromosomes. *NLP* represents the technique proposed in [4] and *NLPCL* denotes our modified approach that considers co-located reward rate and the memory intensity of task types. The results in this section that show error bars represent the 95% confidence intervals around the mean for 24 trials, with each trial varying in the ECS, deadline, reward values, and arrival rate values.

Fig. 2 shows a comparison of the greedy, GA, and NLP techniques evaluated using the reward rate (RR) measure (hashed bars) that does not consider co-location effects in addition to a comparison of all resource allocation techniques evaluated using our new co-located reward rate (CRR) measure (non-hashed bars). We normalize RR and CRR by the same total reward rate that the system is capable of obtaining, i.e., when all task types are executing at rates equal to their arrival rates. To make a fair comparison, the results for the GA and GACL heuristics are recorded at the same amount of heuristic execution time it takes to perform NLP and NLPCL, respectively (about one hour for the small platform size shown). All techniques are able to meet the power and thermal constraints.

It can be observed in Fig. 2 that it is very important to consider the co-location interference, as evidenced by the performance of GACL compared to GA and NLPCL compared to NLP for co-located reward rate. GACL and NLPCL significantly outperform their counterparts for co-located reward rate. By incorporating the effects of co-

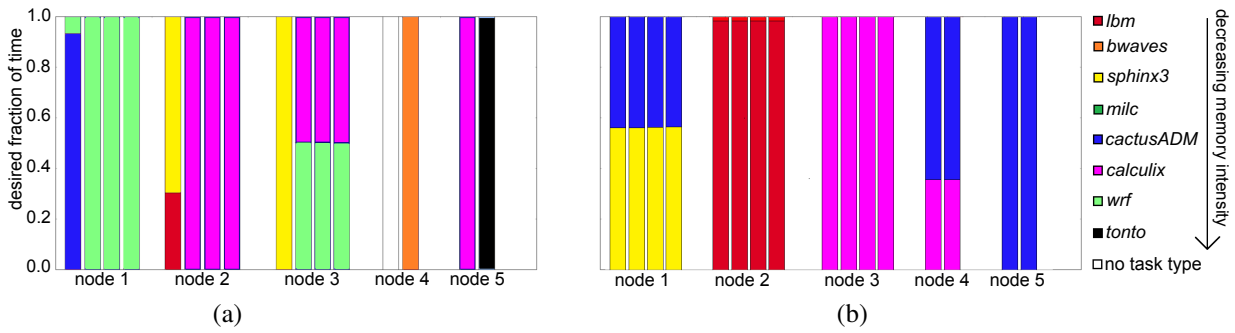

Figure 3. Desired fractions of time for task types allocated to five nodes (16 cores total) on the small platform size when using (a) NLPCL and (b) NLP.

location interference into the fitness evaluation, GACL can make intelligent choices for selecting parent chromosomes that reduce the effects of co-location interference as evidenced by the results for co-located reward rate. Similarly, NLPCL avoids allocations that result in extreme performance degradation, such as placing two memory-intensive task types on cores within the same processor. NLP provides a very poor CRR, but NLPCL provides the best CRR.

Fig. 2 also shows the normalized reward rate (hashed bars) for the co-location unaware greedy, GA, and NLP approaches. The reward rate for these approaches appears to be high compared the results for co-located reward rate. However, in the presence of co-location interference, the normalized reward rate is an inaccurate metric that does not consider performance degradation from co-location interference. Thus, it overestimates the achievable reward rate.

Fig. 3 provides an in-depth view of the actual allocations provided by NLPCL and NLP across five (out of the 150) nodes from Fig. 2. Fig. 3a shows an example of how NLPCL allocates the desired fractions of time in cores within the same processor (i.e., those that share memory resources). In the figure, a bar represents the desired fraction of time a core is allocated to different task types. Nodes 1, 2, and 3 have quad-core processors and nodes 4 and 5 have dual-core processors. Task types in the legend are ordered by their memory-intensity. We can observe in Fig. 3a that NLPCL mitigates the impact of co-location interference by avoiding allocating memory-intensive task types on cores within the same processor. We can see that the processors in nodes 1, 2, and 3 allocate memory-intensive task types to only one core, and allocate CPU-intensive task types to the remaining cores to reduce performance degradation. Interestingly, one of the cores in node 4 is left to idle (no task type assigned) so the memory-intensive task type allocated to the other core in node 4 can execute with no interference.

Fig. 3b shows the desired fractions of time allocations on those same nodes provided by the NLP algorithm that does not consider co-location. In our heterogeneous environment, different task types perform better (i.e., execute faster) on different node types. As it lacks knowledge of co-location interference, NLP typically chooses to allocate task types to nodes that can execute them the fastest. For example, in Fig. 3b, *bwaves* is the primary task type allocated to node 2, and *calculix* is the only task type allocated to node 3, as nodes of these node types were found to be the best-performing (i.e., greatest ECS values) for those task types. Without considering co-location interference this may seem to be a "good" choice, however we can see that nodes 1 and 2 have memory-intensive task types allocated to all cores, which causes severe performance degradation. It is obvious from Figs. 2 and 3 why it is important to consider co-location interference, as NLPCL and GACL significantly outperform the techniques that do not consider co-location.

As HPC facilities become larger, it is important to address the practicality of resource allocation techniques on larger system complexities. Fig. 2a showed that NLPCL was able to achieve the best results for the 150 node (small) platform size. However, the primary drawback associated with the NLPCL technique is its poor scalability. Genetic algorithms hold an advantage in that they are able to provide a solution within any given algorithm runtime bounds, though typically better results are obtained the longer they run. Fig. 4 shows a comparison of greedy, GACL, and NLPCL on the medium platform. We recorded the performance of GACL at many different heuristic execution times to examine the benefits of running it as long as the NLPCL algorithm takes to execute (twelve hours), and as short as the greedy algorithm takes to execute (just over a minute). Even though the GACL algorithm still does not perform as well as NLPCL in the same amount of time, we can obtain a reasonably good solution from GACL whenever one is needed, e.g., for any system administrator specified heuristic runtime bounds. We can also see that GACL does not perform as well as the greedy heuristic in the same amount of time that greedy takes to execute, because GACL is only barely able to generate an initial population in the same amount of time it takes the greedy heuristic to execute.

Fig. 5 compares greedy and GACL at many different heuristic execution time durations. NLPCL results were excluded for the large platform size because we ran the algorithm for a week (168 hours) and it still had not finished. Similar to Fig. 4, greedy is able to outperform GACL

in a similar amount of time, however the hill climbing capabilities of GACL provide better results given more time.

In summary, for a small platform and problem size, the better results obtained using the non-linear programming (NLPCL) approach motivates its use to generate resource allocations that optimize co-located reward rate. However, as the platform size becomes larger, the NLPCL technique becomes intractable, and GACL offers a solution in a reasonable amount of time. If a resource allocation needs to be found quickly, our greedy heuristic may be the desired approach to take. Over time, however, our GACL heuristic is able to provide better solutions and can be terminated at any point, e.g., when the steady-state of the data center changes and new execution rates and P-states need to be found.

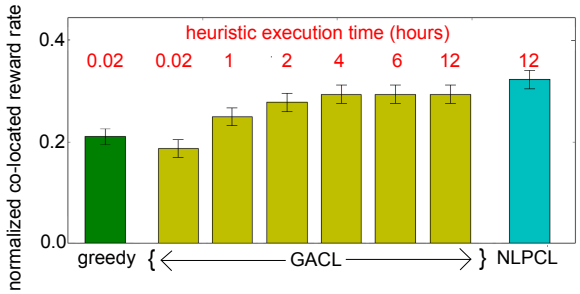

Figure 4. Co-located reward rate of greedy, GACL (across different heuristic execution time durations), and NLPCL on the *medium* platform.

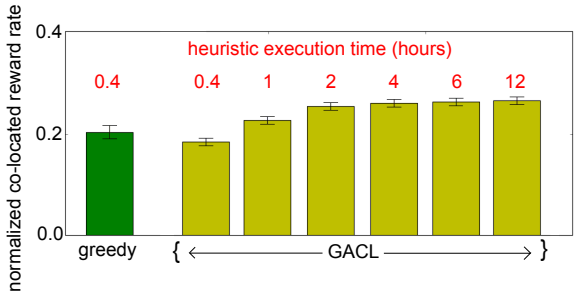

Figure 5. Co-located reward rate of greedy and GACL (across different heuristic execution time durations) on the *large* platform.

VII. Conclusions

We study the problem of maximizing the reward collected for completing tasks by their deadlines subject to power and thermal constraints. Co-location interference can have a significant impact on the execution speeds of tasks (and thus total reward). We capture these effects with our new performance metric called co-location reward rate. We designed: a greedy technique based on assigning tasks to machines in order of their efficiency (most performance per unit of power), a genetic algorithm combined with a local search technique that maximizes the co-located reward rate and ensures the power and thermal constraints are met, and modified a non-linear programming technique from our prior work to maximize co-located reward rate. We show the importance of considering performance degradation due to co-location interference by analyzing the co-located reward earned by our proposed techniques when they consider co-location interference and when they do not consider it.

The results show that NLPCL offers superior solutions to our other proposed techniques, but NLPCL scales poorly and the time to execute the algorithm increases dramatically as the number of nodes and task types increases. When GACL executes for the same amount of time it takes our greedy to execute, the greedy heuristic provides better results as our GACL heuristic because the GACL is not able to execute very many iterations to improve upon the initial (random) population at that time. However, our GACL algorithm provides better solutions over time and can be terminated when a solution is needed.

We have several directions we wish to pursue going forward with this study. We would like to improve our genetic algorithm by initializing the population with different seeds to help guide the search in an intelligent manner, as well as experiment with other methods of constrained optimization besides the local search technique. We are designing a fast greedy linear programming approximation approach to use as a seed and for comparison, and another greedy heuristic that intelligently avoids co-location effects by assigning desired fractions for task types that avoid interference (much like the allocation for NLPCL shown in Fig. 3a). We are also working on creating a more refined model of task memory intensities and co-location interference, and considering a greater variety of workloads for our evaluation. Lastly, we want to perform a sensitivity analysis on how the degree of heterogeneity of the system affects our techniques [25].

Acknowledgments

The authors thank Ryan Friese and Bhavesh Khemka for their valuable comments on this work. This research was supported by NSF grants CNS-0905399, CCF-1302693, and CCF-1252500. This research used the CSU ISTeC Cray System supported by NSF Grant CNS-0923386.

References

[1] W. Feng, "The Green500 list - November 2013," Nov. 2013, accessed 24 Feb. 2014. [Online]. Available: http://www.green500.org/lists/green201311

[2] P. Kogge *et al.*, "Exascale computing study: Technology challenges in achieving exascale systems," DARPA, Tech. Rep., Sep. 2008, http://users.ece.gatech.edu/mrichard/ExascaleComputingStudyReports/exascale_final_report_100208.pdf. Accessed 5 Feb. 2014.

[3] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: Quantifying effects of shared on-chip resource interference for consolidated virtual machines," in *2nd ACM Symp. on Cloud Computing (SOCC '11)*, Oct. 2011, pp. 1–14.

[4] A. M. Al-Qawasmeh, S. Pasricha, A. A. Maciejewski, and H. J. Siegel, "Power and thermal-aware workload allocation in heterogeneous data centers," *IEEE Trans. Computers*, online preprint May 2013, http://doi.ieeecomputersociety.org/10.1109/TC.2013.116.

[5] J. Bruschi, P. Rumsey, R. Anliker, L. Chu, and S. Gregson, US Dept. of Energy, Tech. Rep., Mar. 2011, http://www1.eere.energy.gov/femp/pdfs/eedatacenterbestpractices.pdf. Accessed 10 Oct. 2013.

[6] H. Shamalizadeh, L. Almeida, S. Wan, P. Amaral, S. Fu, and S. Prabh, "Optimized thermal-aware workload distribution considering allocation constraints in data centers," in *IEEE Int'l Conf. on Cyber, Physical and Social Computing (CPSCom '13)*, Aug. 2013, pp. 208–214.

[7] M. Islam, S. Ren, N. Pissinou, H. Mahmud, and A. Vasilakos, "Distributed resource management in data center with temperature constraint," in *4th Int'l Green Computing Conf. (IGCC '13)*, June 2013, pp. 31–40.

[8] J. Patel, S. Guercio, A. Bruno, M. Jones, and T. Furlani, "Implementing green technologies and practices in a high performance computing center," in *4th Int'l Green Computing Conf. (IGCC '13)*, June 2013, pp. 1–8.

[9] A. Banerjee, T. Mukherjee, G. Varsamopoulos, and S. K. S. Gupta, "Cooling-aware and thermal-aware workload placement for green hpc data centers," in *1st Int'l Green Computing Conf. (IGCC '10)*, Aug. 2010, pp. 245–256.

[10] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *44th Int'l Symp. on Microarchitecture (MICRO '11)*, Dec. 2011, pp. 248–259.

[11] M. Caccamo, R. Pellizzoni, L. Sha, G. Yao, and H. Yun, "MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," in *19th Real-Time and Embedded Technology and Applications Symp. (RTAS '13)*, Apr. 2013, pp. 55–64.

[12] Hewlett-Packard, Intel, Microsoft, Phoenix Technologies, and Toshiba, *Advanced Configuration and Power Interface Specification*, 2011, rev. 5.0, http://www.acpi.info. Accessed 28 Aug. 2012.

[13] M. A. Iverson, F. Ozgüner, and L. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," *IEEE Trans. Comput.*, vol. 48, no. 12, pp. 1374–1379, Dec. 1999.

[14] Y. A. Li, J. K. Antonio, H. J. Siegel, M. Tan, and D. W. Watson, "Determining the execution time distribution for a data parallel program in a heterogeneous computing environment," *J. Parallel and Distributed Computing*, vol. 44, no. 1, pp. 33–52, July 1997.

[15] V. Shestak, J. Smith, A. A. Maciejewski, and H. J. Siegel, "Stochastic robustness metric and its use for static resource allocations," *J. Parallel and Distributed Computing*, vol. 68, no. 8, pp. 1157–1173, Aug. 2008.

[16] J. Moore, J. Chase, P. Ranganathan, and R. Sharma, "Making scheduling "cool": Temperature-aware workload placement in data centers," in *USENIX Annual Technical Conf. (ATEC '05)*, Apr. 2005, pp. 61–75.

[17] Q. Tang, T. Mukherjee, S. K. Gupta, and P. Cayton, "Sensor-based fast thermal evaluation model for energy efficient high-performance datacenters," in *4th Int'l Conf. on Intelligent Sensing and Information Processing (ICISIP '06)*, Dec. 2006, pp. 203–208.

[18] D. Dauwe, R. Friese, S. Pasricha, A. A. Maciejewski, G. A. Koenig, and H. J. Siegel, "Modeling the effects on power and performance from memory interference of co-located applications in multicore systems," 2014, 7 pp., *submitted.*

[19] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, R. F. Freund, D. Hensgen, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, June 2001.

[20] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *J. Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107–131, Nov. 1999.

[21] D. Whitley, "The GENITOR algorithm and selective pressure: Why rank-based allocation of reproductive trials is best," in *3rd Int'l Conf. on Genetic Algorithms*, June 1989, pp. 116–121.

[22] Standard Performance Evaluation Corporation (SPEC), 2008, SPECpower_ssj2008, http://www.spec.org/power_ssj2008. Accessed 28 Dec. 2013.

[23] Standard Performance Evaluation Corporation (SPEC), 2006, SPEC_CPU2006, http://www.spec.org/cpu2006. Accessed 2 Jan. 2014.

[24] S.-G. Kim, C. Choi, H. Eom, H. Y. Yeom, and H. Byun, "Energy-centric DVFS controlling method for multi-core platforms," in *5th Int'l Workshop on Multi-Core Computing Systems (MuCoCoS '12)*, June 2012, pp. 685–690.

[25] A. M. Al-Qawasmeh, A. A. Maciejewski, R. G. Roberts, and H. J. Siegel, "Characterizing task-machine affinity in heterogeneous computing environments," in *20th Heterogeneity in Computing Workshop (HCW '11)*, May 2011, pp. 33–43.