Energy-optimal Configuration Selection for Manycore Chips with Variation

Akhil Langer[†], Ehsan Totoni[†], Udatta Palekar[‡] and Laxmikant V. Kalé[*]

[†] *Intel Corporation*
{akhil.langer, ehsan.totoni}@intel.com

[‡] *College of Business,* [*] *Department of Computer Science*
University of Illinois at Urbana-Champaign
{palekar, kale}@illinois.edu

Abstract

Operating chips at high energy-efficiency is one of the major challenges for modern large scale supercomputers. Low voltage operation of transistors increases the energy efficiency but leads to frequency and power variation across cores on the same chip. Finding energy-optimal configurations for such chips is a hard problem. In this work, we study how integer linear programming techniques can be used to obtain energy efficient configurations of chips that have heterogeneous cores. Our proposed methodologies give optimal configurations as compared to competent but sub-optimal heuristics while having negligible timing overhead. The proposed ParSearch method gives up to 13.2% and 7% savings in energy while causing only 2% increase in execution time of two HPC applications - miniMD and Jacobi, respectively. Our results show that integer linear programming can be a very powerful online method to obtain energy-optimal configurations.

Keywords: Energy, Power, Optimization, Multicore chips, Low Voltage Computing, Near Threshold Voltage Computing, Process Variation, Heterogeneity, Integer Programming, Quadratic Integer Programming

1. Introduction

Future microprocessor chips are expected to have variations across the many cores because of the variation in the CMOS manufacturing process. The variation across the chip is expected to further increase with low voltage operation. Chips with low voltage operation have high energy efficiency that is required to build an exascale machine with a power budget of 20MW set by the U.S. Department of Energy. Therefore, it is accepted in the High Performance Computing (HPC) community that there will be heterogeneity across cores of the future generation chips [3]. Frequency and static power consumption of the cores on the same chip can be very different. Low voltage operation can cause up to 50% variation in frequency across the cores of the same chip. Variation across multiple cores on the same chip can also be obtained on the recent Intel® Haswell chip which allows independent core-frequency scaling, that is, various cores on the same chip can be controlled by the user to run at different frequencies. However, unlike the Intel® Haswell chip, heterogeneity across the cores in a chip with low voltage operation will be forced on to the user (unless all the cores are made to run at a minimum frequency, which will be a very inefficient design).

Energy is one of the biggest challenges faced by the HPC community. Data centers worldwide consumed energy equivalent to 235 billion KWh in 2010, which is 2% of total US electricity consumption. CPU accounts for about 65% of the total power consumption of a supercomputer [55]. Therefore, minimizing CPU energy consumption is critical for overall savings in the energy costs of a data center. In this work, we focus on intelligent selection of the cores on a chip with variation for running parallel HPC applications such that the energy consumption of the chip is minimized given execution time constraints. To provide the necessary background, we discuss the variation in future chips, the programming systems that can mitigate the impact of heterogeneity on application performance, and performance modeling of the chip in the wake of heterogeneity. We formulate the energy minimization problem with constraints on execution time as a constrained optimization problem. The problem is a cubic integer programming problem and is hard to solve. This paper extends the material presented in our previous publication [35] by developing new methods that give competitive configurations with very high accuracy while taking magnitudes of order lesser time to obtain the solution. We propose two methods namely the TransformedQP [35] and the ParSearch method to solve this problem. We show that while the former method gives optimal configurations, it takes more time to obtain the solution. On the other hand, ParSearch method gives near-optimal configurations (that are within 99.8% of the optimal configuration in energy efficiency) and has very small overhead as compared to TransformedQP method. We also propose a very fast GoodCores heuristic that gives competitive configurations in cases when the allowed execution time penalty on the application is high. Our results show that intelligent selection of cores using integer programming can lead to significant savings in energy costs with very small increase in execution time, while incurring negligible overhead to obtain the solution.

The paper is divided into 7 sections. In Section 2, we perform a survey of literature on energy optimizations for HPC workloads. In Section 3, we discuss process variation that leads to heterogeneity across the cores on a chip, and performance modeling for such chips. The proposed methods, TransformedQP and ParSearch, for energy minimization are discussed in Section 4. The evaluation setup is given in Section 5, which is followed by results and their analysis in Section 6. Finally, we conclude the paper with conclusions and future work in Section 7.

2. Related Work

In the past, the emphasis has been on minimizing the completion times of the HPC applications. However, such solutions could have excessive energy consumption and hence high energy costs for the data centers. Consequently, minimizing energy consumption has become a major challenge for high performance computing data centers, especially with the increase in the size of the data centers. Hence, minimizing energy consumption has been a subject of intensive research over the past several years.

Dynamic power consumption of a chip is known to be a function of the frequency of the chip [23]. Applications do not yield proportionate improvement in performance with the increase in frequency of the chip, and therefore frequency is scaled down to reduce the power consumption of the chip while having tolerable impact on the completion time of the application. Modern processor architectures allow users to control the frequency of the chip through DVFS modules. There have been many studies on the use of DVFS for energy efficient computing for HPC [44, 38, 48, 27] and multicore [8] workloads. Rizvandi et al [42] make some observations on optimal frequency selection in DVFS-based energy consumption minimization. Etinski et al [20] present a model that predicts the upper bound on performance loss due to frequency scaling. They study how sensitivity of the application to frequency scaling together with cluster characteristics determines the effectiveness of DVFS for energy consumption optimization. Wang et al [56] propose an energy aware scheduling heuristic that studies the slack time of non-critical tasks, and extends their execution time (by using DVFS) to save energy without affecting the overall execution time of the job. Alonso et al [9] propose methods of improving power efficiency of dense linear algebra algorithms on multi-core processors using slack control. Vishnu et al [54] leverage DVFS to use the slack in one-sided communication primitives of PGAS for energy efficiency.

Lower core frequency also leads to lower core temperatures. DVFS has also been used for controlling the temperature of the chips, which reduces the temperature of the hot spots, that is, the nodes with highest temperature in the data center. Lower temperature of the hot spots means reduction in the cooling energy required to keep the temperatures of the hot spots at the room temperature, thereby reducing the cooling energy costs of the data center. There has been a significant amount of work on various strategies for reducing the cooling energy of HPC and non-HPC data centers [10, 50, 41, 11, 57, 58, 49].

Energy efficiency has been studied extensively in the context of large scale cloud computing as well [59, 52]. The richness of the literature on energy optimization for data centers establishes the importance of this work.

Recent processor architectures, such as IBM Power6 [13], IBM Power7 [15], AMD Bulldozer [4], Intel® Sandybridge [43], provide the user with the ability to control the power consumption of CPU, DRAM, etc. The ability to constrain the power consumption of nodes provides the flexibility to add more nodes to the data center while remaining within the same power budget. This is also called overprovisioning. In our previous work ([45, 46]), we have shown significant improvement in performance of a data center by using overprovisioning under a strict power budget. We have also shown the benefit of using integer linear programming methods for improving the performance of applications on chips with low voltage operation under a strict power budget [51]. In contrast, the focus of this work is on minimizing the energy consumption of the chips, which is an even harder problem to solve because of the cubic and quadratic terms involved in the formulation of the problem.

Previous work (e.g. [33]) has proposed heterogeneous chip designs that have custom designed cores for a given set of target workloads. Different cores are designed to cater to different classes of applications. On the contrary, heterogeneity in the low voltage chips is inherent in the manufacturing process. Integer linear programming has been used in the past in the context of homogeneous multiprocessor chips. Kadayif et al [28] use integer linear programming for determining the optimal number of cores that will be used in executing each nest in the code of array-intensive applications under energy and performance constraints. Power Aware Resource Manager, PARM, proposed by Sarood et al [45] uses Integer Linear Program (ILP) to schedule and determine the optimal allocation of power and compute nodes to jobs submitted to a data center. Venugopalan et al [53] propose the use of ILP for optimal task scheduling on multiprocessors. To the best extent of our knowledge, energy efficiency in the context of chips with low voltage operation has not been addressed before.

3. Preliminaries

In this section, we review the causes of heterogeneity for future generation chips, and its impact on performance of parallel applications, such as load imbalance. We then briefly study some of the programming systems that can overcome the impact on applications performance by performing load balancing of over-decomposed tasks. Finally we discuss some performance models that can predict an application's performance in such a heterogeneous environment. The performance models will be used in the next section (Section 4) for optimal selection of cores for energy-efficient computing. More details about these preliminaries can be found in previous work [51].

3.1. Process Variation

Operating at low voltage leads to increase in energy-efficiency of the chip. High energy efficiency of operation at low voltages has been established for 65, 45, 32, 22 nm technologies [32, 7, 25, 31]. Kaul et al [32, 30] show that as the supply voltage of the transistor is reduced, the energy efficiency increases, and is maximum near the threshold voltage of the transistor. At threshold voltage, energy efficiency is $10\times$ as compared to at the nominal supply voltage. However, as the supply voltage reaches near the threshold voltage, even a small change in the supply voltage leads to large spread in the frequency of operation. Therefore, different cores will be operating at different frequencies in a manycore chip. Leakage power also varies significantly across chips. More challenges associated with low voltage operation can be found in [30]. Nearest frequency of operation is assigned to these cores as shown in Figure 1.

3.2. Programming Systems

HPC applications are highly synchronized applications. For example, in many applications all the processors synchronize after every iteration (or every few iterations) to exchange neighbor boundaries. Hence, the speed of execution of a parallel HPC application is only as fast as the speed of the slowest processor. Of course, this is true only if the workload is distributed equally to the processors. When the processors have different speeds, the work load assigned to a core should be proportionate to its speed of operation. In order to do so, the total work has to be over-decomposed into many small tasks (more than the number of cores), such that it can be evenly distributed to the cores in proportion to their frequencies. It is not always possible

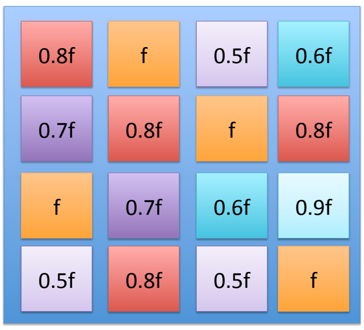


Figure 1: An example of core frequencies in a manycore chip with variation across cores.

to ensure load balance in such a situation. For example, if there are two processors with frequencies f and 0.75f, and there are three equal sized tasks, then it is impossible to achieve perfect load balance. However, as the total number of tasks increases, the load imbalance decreases (provided an intelligent algorithm for load distribution is being used). Previous work [51] has shown that with an over-decomposition level of 16 (that is, the number of tasks to number of cores ratio is 16), the load imbalance can be contained to within 2-6% of the total execution time of the application. There are many parallel programming languages that over-decompose the total work into many small tasks. Some examples of such distributed parallel programming languages are Charm++ [6], AMPI [26], etc. For shared memory machines, Cilk [14], OpenMP [19], etc. are some examples of programming models in which the work is divided into chunks (for example, iterations in for loops in OpenMP) that can be dynamically assigned to processors during runtime.

For our proposed method, no changes are required either in the programming language or in the code (except possibly the addition/use of a variation aware load balancer).

3.3. Performance Modeling

In this section, we discuss the models to predict the performance (execution time or instructions per cycle) of a parallel HPC application on any configuration of a heterogeneous manycore chip. A configuration is a subset of the cores on the chip on which the parallel application will be executed. Other cores on the chip are turned-off so that they do not consume any static power. A good configuration of the chip for a given HPC application minimizes the total energy consumption during the execution period of the application. It is practically infeasible to evaluate all possible configurations of the chip for every application because the total number of configurations is combinatorially large. For example, when the number of cores on the chip is 36, the total number of possible configurations is $2^{36} - 1 \approx 6.87e10$. Therefore, performance models are required that can predict the performance of an application for any configuration. The model should require minimal profiling information of the application to be collected, so that the overhead of developing the performance models is negligible.

We now review the performance models for manycore heterogeneous chips from previous work [51]:
Model 1: All the cores can be individually profiled for the application, and the performance for a given configuration could be modeled as the sum of the performance of the individual cores in the configuration (c).

$$S = \sum_{i \in c} s_i \tag{1}$$

where, s_i is the performance (instructions per cycle) of core i for the focal application when the application was run only on core i, and S is the predicted performance (instructions per cycle) for configuration c for the focal application. This model will predict performance accurately only for computationally intensive applications in which there is no memory contention. For memory-intensive applications, this performance model will fail to predict the performance for a configuration because it just adds the core performance which was obtained when they were running individually, and does not model the contention for the shared resources, e.g. memory, when multiple cores are running simultaneously.

Model 2: The application execution time is divided into two components: T_{cpu} corresponding to CPU time and T_{mem} corresponding to memory time (as in [24, 18, 17, 47]). And the performance is modeled as

$$T = \frac{T_{cpu}}{\sum_{i \in c} f_i} + T_{mem} \tag{2}$$

where, f_i is the frequency of core i, and T is the predicted execution time of the application. The weakness of this model is that it fails to incorporate the number of cores that are accessing the memory, and treats the memory time as constant irrespective of the cores that are accessing the memory.

Model 3: In this model, we construct as many model functions as there are number of cores on the chip. There is one model for all the configurations with the same number of cores. For instance, if there are 36 cores on a chip, 36 functions are developed. In this way, this model incorporates the number of active cores in performance prediction. Each of these functions is a linear function of the sum of frequencies of the cores in the configuration. The performance (instructions per cycle) function for all the configurations with k cores is modeled as:

$$S = a_k(\sum_{i \in c} f_i) + b_k \tag{3}$$

where, a_k, b_k are line constants for all configurations with k cores, and S is the instructions per cycle of the configuration. Only two performance data samples are required to get the value of the constants, a_k and b_k, for this function. These samples correspond to instructions per cycle for any two configurations with k cores. Since there are n functions, $2n$ samples are sufficient to develop the complete model for an application (although more samples can increase the accuracy of the model). The overhead of sampling the data to generate the model is negligible as compared to the execution time of HPC applications, which can be from hours to days. In previous work [51], it is shown that the prediction accuracy of Model 3 is very high. The average prediction error in performance is less than 1.6%, and 0.7% for a computationally intensive and a memory intensive application, respectively. Simulated performance was obtained using the Sniper simulator, discussed in detail in Section 5. Similar to performance, the dynamic power consumption of a configuration could be modeled accurately using Model 3, that is,

$$P = A_k(\sum_{i \in c} f_i) + B_k$$

where P is the dynamic power of configuration c, A_k and B_k are line constants. It has been shown in previous work [51] that the maximum prediction error of Model 3 for dynamic power is less than 2%.

4. Energy Optimization Approach

In this section, we describe our approach for optimizing the energy consumption during application execution. The total energy is computed as the power consumption integrated over the duration of execution of the application, that is, power consumption multiplied by the execution time of the application. We use Model 3, described in the previous section, to model the execution time and dynamic power consumption of any configuration. According to Model 3, the linear function for performance and dynamic power consumption of a configuration depends on the number of cores in the configuration. Therefore, the energy consumption can be defined as

$$\sum_{k=1}^{N} (n_k * (a_k^p \sum_i x_i f_i + b_k^p + \sum_i s_i x_i) * (a_k^t \sum_i x_i f_i + b_k^t))$$

where, n_k is a binary variable indicating whether the selected configuration has k cores (n_k can be 1 only for one value of k) , x_i is a binary variable indicating whether i^{th} core is selected, $a_k^p \sum_i x_i f_i + b_k^p$ is the

Table 1: Constrained Optimization Program Terminology

Symbol	**Description**
n	total number of cores on the chip
c	a configuration
k	an index variable used to represent number of cores in a configuration
n_k	binary variable indicating whether the selected configuration has k cores
x_i	a binary variable indicating whether core i is selected or not in a configuration
f_i	frequency of core i
f	a variable that equals the sum of the frequencies of the cores in the selected configuration, that is, $f = \sum_i x_i f_i$
s_i	static power consumption of core i
a_k^t, b_k^t	line constants for performance model of configurations with k cores
a_k^p, b_k^p	line constants for dynamic power model of configurations with k cores
t_{min}	minimum execution time of the application across all the configurations on the chip
tp	penalty in execution time, maximum allowed execution time is $(1 + \frac{tp}{100}) \times t_{min}$

dynamic power consumption of the configuration, s_i is the static power consumption of core i, $\sum_i s_i x_i$ is the total static power consumption, and $a_k^t \sum_i x_i f_i + b_k^t$ is the execution time of the application. The energy minimization problem can then be formulated as a constrained optimization problem. The formulation is given in Equations (4)-(8) (Program 1). Terminology used in this section is defined in Table 1.

Objective Function

$$min \sum_{k=1}^{n} n_k * (a_k^p \sum_{i=0}^{n-1} x_i f_i + b_k^p + \sum_{i=0}^{n-1} s_i x_i) * (a_k^t \sum_{i=0}^{n-1} x_i f_i + b_k^t) \qquad (4)$$

Select One Value of k

$$\sum_{k=1}^{n} n_k = 1 \qquad (5)$$

Total Number of Cores Equals k

$$\sum_{i=0}^{n-1} x_i = \sum_{k=1}^{n} n_k k \qquad (6)$$

Variables Range

$$\forall i \in [0, n), \quad x_i \in \{0, 1\} \qquad (7)$$

$$\forall k \in (0, n], \quad n_k \in \{0, 1\} \qquad (8)$$

Program 1: Original optimization program for minimizing energy consumption. The objective function has cubic terms.

Constraints in the above formulation are linear constraints that ensure that a valid configuration is selected. However, the objective function has a cubic expression. This constrained optimization problem can be readily solved by solving n quadratic integer programs. Each of these quadratic integer programs chooses the best configuration amongst all the configurations with the same number of cores. The best performing configuration is then chosen from amongst the optimal configurations returned by the n quadratic integer program optimizations. In this way, the global optimal configuration can be found by optimizing n quadratic programs (Algorithm 1). The quadratic program that selects the best configuration from amongst all the configurations with k cores is given below in Equations (9)-(11) (Program 2).

Algorithm 1 Algorithm for obtaining the globally optimal configuration by solving n quadratic programs

```
for $k \in [1, n]$:
  //Obtain the best configuration amongst all configurations with $k$ cores
  $C_k$ = EnergyQP($k$)

//energy($C_K$) is the total energy consumption of configuration $C_k$
Optimal Configuration = $\{C_k | energy(C_k)$ is minimum for $k \in [1, n]\}$
```

Objective Function

$$min \quad (a^p_K \sum_{i=0}^{n-1} x_i f_i + b^p_K + \sum_{i=0}^{n-1} s_i x_i) * (a^t_K \sum_{i=0}^{n-1} x_i f_i + b^t_K) \tag{9}$$

Total Number of Cores Equals K

$$\sum_{i=0}^{n-1} x_i = K \tag{10}$$

Variables Range

$$\forall i \in [0, n), \quad x_i \in \{0, 1\} \tag{11}$$

Program 2: Program for finding the minimum energy configuration from amongst all configurations with K cores. The objective function has quadratic terms.

Quadratic programs must have positive semi-definite matrices to be solved using convex optimization. The resulting quadratic programs above are not positive semi-definite and hence can be computationally very hard to solve using non-linear optimization methodologies. We now describe two different methods, namely the TransformedQP and the ParSearch method for solving this constrained optimization problem.

4.1. TransformedQP *Method*

In order to reduce the quadratic objective function to a linear expression, we use the scheme proposed by Glover and Woosley [22]. In this scheme, the cross-product terms in the objective function are replaced by adding new continuous variables. The value of these new variables are determined by adding new constraints. For example, a quadratic product term $x_1 x_2$, where x_1, x_2 are binary variables, can be replaced by a new variable y_{12} such that $y_{12} \leq x_1, y_{12} \leq x_2,$ and $y_{12} \geq x_1 + x_2 - 1$. We multiply the terms in the objective function (Equation 9) and replace the product terms of the form $x_i x_j$ with new continuous variables y_{ij}. The resulting ILP is given below in Equations (12)-(15) (Program 3).

This transformation from the quadratic program to linear program increases the number of variables from v to $\frac{v(v+1)}{2}$, and the number of constraints from 1 to $\frac{3v(v-1)}{2}$. Since the value of v is small for the focal problem, the size of the resulting integer linear program remains tractable.

Objective Function

$$\begin{aligned} min \quad \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}(a_K^p f_i + s_i)(a_K^t f_j) y_{ij} + \; b_K^t \sum_{i=0}^{n-1}(a_K^p f_i + s_i) x_i \\ + b_K^p a_K^t \sum_{j=0}^{n-1} f_j x_j + b_K^p b_K^t \end{aligned} \tag{12}$$

Total Number of Cores Equals K

$$\sum_{i=0}^{n-1} x_i = K \tag{13}$$

New variable constraints

$$\begin{aligned} y_{ij} &\le x_i, & \forall i, j \in [0, n), j \le i \\ y_{ij} &\le x_j, & \forall i, j \in [0, n), j \le i \\ y_{ij} &\ge x_i + x_j - 1, & \forall i, j \in [0, n), j \le i \end{aligned} \tag{14}$$

Variables Range

$$\forall i \in [0, n), \quad x_i \in \{0, 1\} \tag{15}$$

Program 3: Quadratic program in Program 2 transformed into an integer linear program by introducing new variables and constraints. We call this as TransformedQP.

It is possible that the configuration with minimum energy consumption has a very large execution time as compared to the best execution time. In order to constrain the increase in execution time, the following time constraint is added to the linear programs, where *tp* is the allowed percentage increase in execution time.

$$a_K^t(\sum_{i=0}^{n-1} x_i f_i) + b_K^t \le (1 + \frac{tp}{100}) * t_{min} \tag{16}$$

This proposed ILP methodology is evaluated in Section 6.

4.2. ParSearch *method*

In the previous section (TransformedQP method), we proposed a methodology to transform the quadratic program (Equation 9-11, Program 2) into an integer program by introducing additional variables and constraints. This leads to increase in the number of variables and constraints to $O(v^2)$, where v is the number of variables in the original quadratic program. In this section, we propose an alternative approach in which we rewrite the quadratic objective function of Program 2 as

$$\min \quad (a_K^p f + b_K^p + \sum_{i=0}^{n-1} s_i x_i) * \alpha t_{min}$$

subject to $a_K^t f + b_K^t = \alpha t_{min}, \alpha \ge 1$. The resulting problem is a linear program for a fixed value of α. Since α is fixed, the resulting linear program has the same solution if the objective function, which represents energy used is replaced with the power function

$$\min \quad a^p_K f + b^p_K + \sum_{i=0}^{n-1} s_i x_i$$

As the value of α is parametrically varied, the resulting minimum power can be shown to be a non-increasing piecewise linear function of α [12]. Determining the breakpoints of this piecewise linear function can be done by using parametric linear programming.

To obtain the minimum energy consumption however, we are interested in $\alpha * t_{min} * power$. While power is a decreasing linear function of α between breakpoints, it is easy to see that energy, which is a quadratic function of α, is concave. If we relax the integrality restrictions, the concavity of the energy function between every pair of breakpoints implies that the minimum will occur at one of the breakpoints. Because of the integrality requirements, however, only some discrete values of α can have feasible solutions. Suppose $\alpha_1, \alpha_2, ..., \alpha_m$ are the feasible discrete values of α between the pair of breakpoints, then the concavity of the energy function means that the minimum energy can only occur at either α_1 or α_m. Program 4 is the basic program for optimization in this approach, and the problem is to search for the optimal value of α ($1 \leq \alpha \leq 1 + 0.01 * tp$), such that the energy consumption is minimized. Selecting the minimum energy consumption between each pair of break points and subsequently selecting the minimum amongst these solutions guarantees an optimal solution. However, this can be time consuming and so we consider an alternative simpler sampling method for determining the optimal values of alpha. In Section 6, we describe the method we used for finding the optimal value of α.

Objective Function

$$min \quad a^p_K \sum_{i=0}^{n-1} x_i f_i + b^p_K + \sum_{i=0}^{n-1} s_i x_i \tag{17}$$

Total Number of Cores Equals K

$$\sum_{i=0}^{n-1} x_i = K \tag{18}$$

Bounding the execution time

$$a^t_K \sum_{i=0}^{n-1} x_i f_i + b^t_K \leq \alpha t_{min} \tag{19}$$

Variables Range

$$\forall i \in [0, n), \quad x_i \in \{0, 1\} \tag{20}$$

Program 4: Basic linear program for optimization in ParSearch method. This integer linear program minimizes power consumption given execution time constraint.

5. The Setup

We use the Sniper Multi-core Simulator [16] for simulating chips with heterogeneity. We use the default core model of Sniper. The default core model is similar to Intel® Gainestown model and has been validated [16]. We simulate chips with 36 cores. Each chip has x86 cores with 4-wide out-of-order issues. Each core has a private 4-way L1 Instruction cache of size 32 KB, and a private 8-way L2 cache of size 256

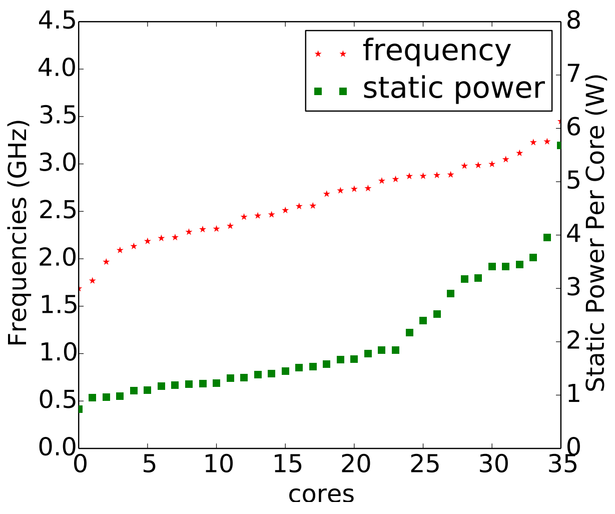


Figure 2: Frequencies and static powers of various cores on a manycore chip with variation.

KB. There is a shared 4-way L1 Instruction cache of size 32KB. The memory latency is 75ns, when there is no memory contention. We use 11-nm technology, with average frequency of 2.6GHz, and V_{dd} of 0.765V. McPAT (Multicore Power, Area, and Timing) [37] framework was integrated with Sniper for dynamic power modeling of manycore applications. Our experimental results in this paper are for chips with 36 cores. For modeling process variation at micro-architectural level and for static power modeling of cores, we use VariusNTV [29]. 25 chips were generated with different core frequencies and static power consumption. The frequencies and the corresponding static power consumption of various cores for one of the chips is shown in Figure 2.

5.1. Applications

Two HPC applications are used for benchmarking the performance:

- *miniMD*: It is a simple, parallel molecular dynamics code that is a micro-application in the Mantevo project at Sandia National Laboratories [1]. miniMD is written in MPI and performs parallel molecular dynamics simulation of a Lennard-Jones system. miniMD is a computationally intensive application.
- *Jacobi*: Jacobi is a 3D stencil computation code. It is a memory intensive application. We use a Charm++ implementation of Jacobi.

Most other HPC applications fall in between miniMD and Jacobi in terms of their computation and memory sensitivity. For developing the performance and dynamic power consumption models of these applications, we obtained $2n$ samples from each application, where n is the number of cores on the chip. For each value of $k \in [1, n]$, 2 samples are required to build the linear model for all configurations with k cores. We chose the configurations with k minimum, and k maximum frequency chips, and obtained their simulated performance on the Sniper simulator. The simulated performance was used to compute the line constants for the performance and power consumption models.

We use the reference energy consumption (*refenergy*) as the energy consumption corresponding to the configuration with the best possible execution time (t_{min}), computed using Algorithm 2. Our results are compared against *refenergy*.

[1]`http://software.sandia.gov/mantevo/`

Algorithm 2 Algorithm for computing the best possible execution time (t_{min}) for an application on the chip

```
1 $t_{min} = 0$
2 for $k \in [1, n]$ :
3   $C_k$ = {cores with $k$ largest frequencies}
4   $t_{min} = \min(t_{min}, a_k^t \sum_{i \in C_k} f_i + b_k^t)$
5 return tmin
```

5.2. Heuristics for Configuration Selection

The proposed integer linear programming approach for energy minimization is compared against three heuristics, called the MinFreq, MaxFreq heuristics, and GoodCores heuristic as described below:

- MinFreq heuristic: The cores are sorted in the increasing order of their frequencies, such that, $f_0 < f_1 < f_2 ... < f_{n-1}$. The heuristic selects the value of k such that the configuration with k consecutive cores, starting from core$_0$ has the minimum energy consumption and the execution time is within the desirable threshold (*tp*). The MinFreq heuristic algorithm is given in Algorithm 3.

Algorithm 3 Algorithm for MinFreq heuristic

```
1 sort frequencies such that $f_0 < f_1 ... < f_{n-1}$
2 energy$_{min}$ = refenergy
3 for $k \in [1, n]$:
4   $C_k$ ={core$_i$ for $i \in [0, k-1]$}
5   time = $a_k^t \sum_{i \in C_k} f_i + b_k^t$
6   if energy($C_k$) < energy$_{min}$ and time < $(1 + \frac{tp}{100}) t_{min}$:
7     energy$_{min}$ = energy($C_k$)
8 return energy$_{min}$
```

- MaxFreq heuristic: The cores are sorted in the decreasing order of their frequencies, such that, $f_0 > f_1 ..... > f_{n-1}$. The heuristic selects the value of k such that the configuration with k consecutive cores, starting from core$_0$ has the minimum energy consumption and the execution time is within the desirable threshold (*tp*). The MaxFreq heuristic algorithm is given in Algorithm 4.

Algorithm 4 Algorithm for MaxFreq heuristic

```
1 sort frequencies such that $f_0 > f_1 ... > f_{n-1}$
2 energy$_{min}$ = refenergy
3 for $k \in [1, n]$:
4   $C_k$ ={core$_i$ for $i \in [0, k-1]$}
5   time = $a_k^t \sum_{i \in C_k} f_i + b_k^t$
6   if energy($C_k$) < energy$_{min}$ and time < $(1 + \frac{tp}{100}) t_{min}$:
7     energy$_{min}$ = energy($C_k$)
8 return energy$_{min}$
```

- GoodCores heuristic: A core that has high frequency for low static power consumption is a good candidate core for selection. Therefore, in this heuristic, the cores are sorted by their value of $\frac{f_i}{s_i}$, such that, $\frac{f_0}{s_0} > \frac{f_1}{s_1}..... > \frac{f_{n-1}}{s_{n-1}}$. The heuristic selects the value of k such that the configuration with k consecutive cores, starting from core$_0$ has the minimum energy consumption and the execution time is within the desirable threshold (tp). The GoodCores heuristic algorithm is given in Algorithm 5.

Algorithm 5 Algorithm for GoodCores heuristic

```
1 sort frequencies such that $\frac{f_0}{s_0} > \frac{f_1}{s_1}..... > \frac{f_{n-1}}{s_{n-1}}$
2 energy$_{min}$ = *refenergy*
3 **for** $k \in [1, n]$:
4   $C_k$ ={core$_i$ **for** $i \in [0, k-1]$}
5   time = $a^t_k \sum_{i \in C_k} f_i + b^t_k$
6   **if** energy($C_k$) < energy$_{min}$ **and** time < $(1 + \frac{tp}{100})t_{min}$:
7     energy$_{min}$ = energy($C_k$)
8 **return** energy$_{min}$
```

5.3. ILP Solver

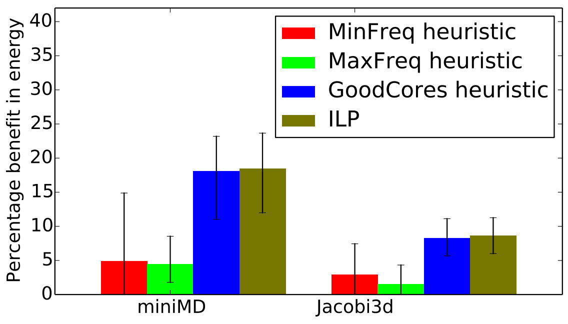
There are several solvers available for integer linear program optimization, such as, Gurobi [1], CPLEX [2], GLPK [40], CBC [21], SCIP [5], Xpress [36]. We use the commercial state-of-the-art solver, Gurobi, for solving the Integer Linear Programs (ILPs). ILPs are NP-hard problems and are solved by using variants and extensions of Branch-and-Bound (BnB) method. In BnB method, the corresponding linear program, obtained by relaxing the integrality constraints on integer variables, is first solved by using the simplex or the interior point method. This gives a fractional solution. Branching is done on the fractional values, which gives more linear programs. Linear program optimizations are done and the branching is continued until an integer solution is found. The integer solution with the best cost acts as an incumbent solution and is used to prune other vertices of the BnB tree that can provably be shown to not have better cost than the current incumbent. Commercial state-of-the art solvers like Gurobi have highly optimized implementations for solving ILPs. They fully exploit the latest mathematical and engineering improvements in the underlying methodologies to provide very fast solutions to linear/mixed-integer programs. Solvers like Gurobi are used for variety of cost and quality optimization purposes in various fields of optimization.

6. Results

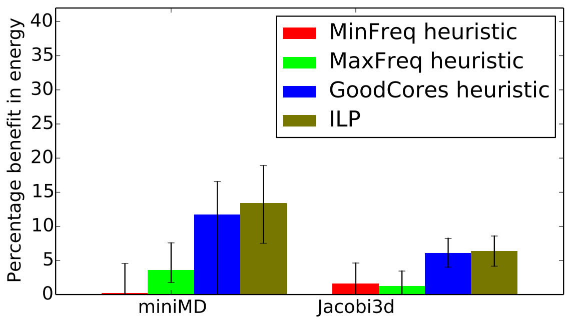
In this section, we first discuss the energy savings obtained using the ILP methodology (the TransformedQP method), and compare it with heuristics. We then compare the ParSearch method with the TransformedQP method in terms of the solution quality and the optimization times to obtain the solution.

6.1. Energy-efficiency

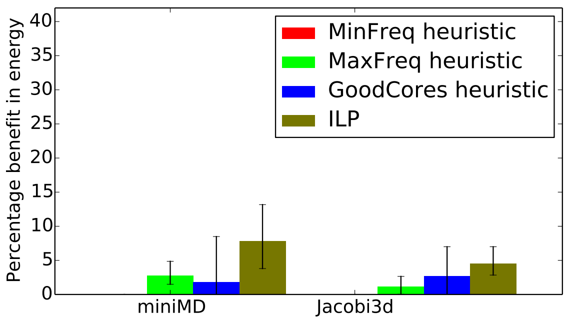
Figure 3 shows the savings in energy consumption by using configurations selected by the MinFreq heuristic, MaxFreq heuristic, GoodCores heuristic and the TransformedQP integer linear programming method when compared to the configuration with best execution time ($energyref$). The results are summary of benefits across 25 different chips. We consider four cases, corresponding to Figure 3a, 3b, 3c, respectively.



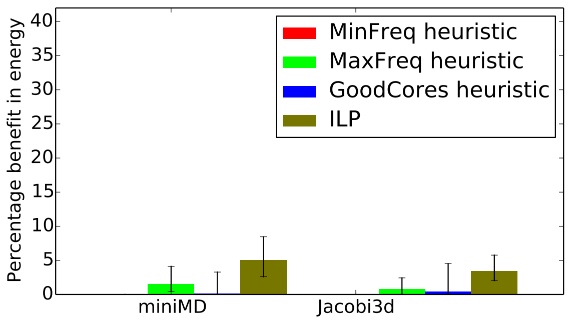
(a) Maximum 15% time penalty



(b) Maximum 5% time penalty



(c) Maximum 2% time penalty



(d) Maximum 1% time penalty

Figure 3: Percentage savings in energy with MinFreq, MaxFreq, GoodCores heuristics, and the TransformedQP ILP method for the two applications, miniMD and Jacobi3d, with respect to the configuration with best execution time. The bars correspond to the average benefits, while the vertical lines correspond to the minimum and maximum benefits obtained from the corresponding method across the 25 chips. In (a), (b), (c), and (d) configuration that minimizes energy consumption while the execution time penalty is less than 15%, 5%, 2%, 1%, respectively, is sought using the various heuristics and the proposed TransformedQP ILP method. While the GoodCores heuristic gives competitive configurations when the time penalty is high, the ILP method performs significantly better when the time penalty is low.

1. In this case, configuration with minimum energy consumption is sought with 15% as the allowed increase in execution time (i.e. $tp = 15\%$).
    - **miniMD** The ILP method gives an average of 18.4% in energy savings, while the savings were 4.9%, 4.4%, and 18% from MinFreq, MaxFreq, and GoodCores heuristics, respectively.
    - **Jacobi** We obtain an average of 8.6% savings in energy consumption using the ILP method. On the other hand, MinFreq, MaxFreq, and GoodCores heuristics give energy savings of 2.94%, 1.5%, and 8.25%, respectively.

    Although we get significant savings in energy by choosing the right configurations, 15% can sometimes considered to a large increase in execution time. Therefore, we consider the following three cases in which there is a very small increase in execution time.
2. When the execution time is less than $1.05 \times t_{min}$, that is, $tp = 5\%$
    - **miniMD** We obtain an average of 0.18%, 3.55%, 11.6%, 13.4% savings in energy with MinFreq, MaxFreq, GoodCores heuristic, ILP, respectively.
    - **Jacobi** An average of 1.5%, 1.25%, 6.07%, 6.4% savings in energy with MinFreq, MaxFreq, GoodCores heuristic, ILP, respectively is obtained.
3. When the execution time is less than $1.02 \times t_{min}$, that is, $tp = 2\%$
    - **miniMD** We obtain an average of 0%, 2.76%, 1.8%, 7.8% savings in energy with MinFreq, MaxFreq, GoodCores heuristic, ILP, respectively.
    - **Jacobi** An average of 0%, 1.12%, 2.6%, 4.6% savings in energy with MinFreq, MaxFreq, GoodCores heuristic, ILP, respectively is achieved.
4. When the execution time is less than $1.01 \times t_{min}$, that is, $tp = 1\%$
    - **miniMD** We obtain an average of 0%, 1.5%, 0.13%, 5.04% savings in energy with MinFreq, MaxFreq, GoodCores heuristic, ILP, respectively.
    - **Jacobi** An average of 0%, 0.78%, 0.4%, 3.43% savings in energy with MinFreq, MaxFreq, GoodCores heuristic, ILP, respectively is achieved.

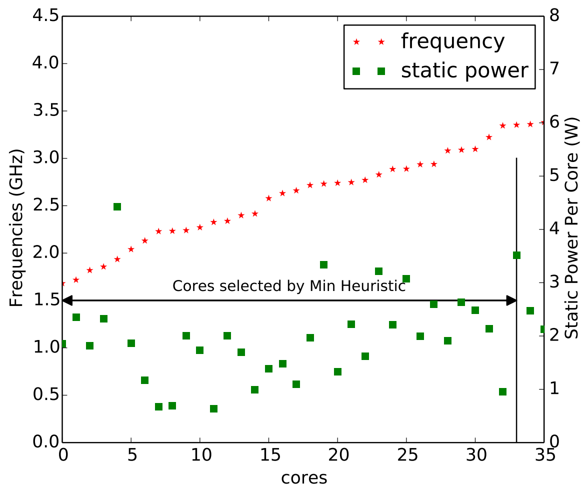
When the timing penalty is high (for example, when tp=15%), the GoodCores heuristic gives comparable results as the ILP. On the other hand for lower timing penalties, the ILP method performs significantly better than any of the sub-optimal heuristics.

Since miniMD is a computationally intensive application, the number of cores in the optimal configuration selected for miniMD are more than the number of cores in the optimal configuration for Jacobi. In Jacobi, large number of cores lead to increase in the memory contention and hence are sub-optimal. Figure 4 shows an example solution obtained from ILP optimization, MinFreq heuristic, and MaxFreq heuristic.

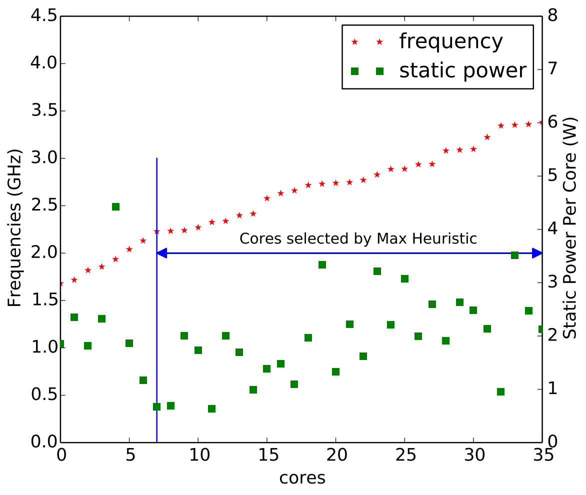
6.2. *Evaluating* ParSearch *Method*

In this section, we present the evaluation of the ParSearch method and compare it with the TransformedQP method. Figure 5 shows the energy value obtained by optimizing the ParSearch ILP for different values of α and K in Program 4. 1000 values of α were sampled between 1 and 1.9. For each K, there is threshold value of α below which no valid configuration can be obtained that satisfies Program 4 and hence there are missing dots in the figure for different values of K. The overall pattern of energy value for all the values of K is that it first decreases (with small aberrations) and then becomes stable with increase in the value of α. Therefore, instead of using the more complicated methods of parametric linear programming, we show that a good value of α can be obtained in a simpler manner through sampling and that value of α gives near-optimal configurations. Fixed number of values of α are obtained by uniform sampling between 1 and the allowed maximum time penalty (tp%), that is,

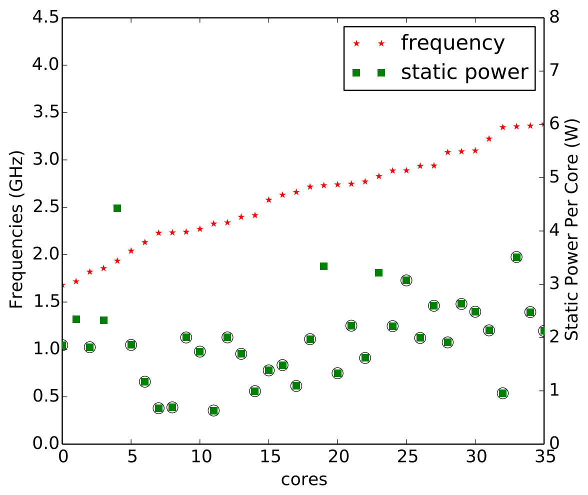
$$\alpha = \{1 + \frac{i * 0.01tp}{nsamples}, \quad \forall i \in [1, nsamples]\},$$



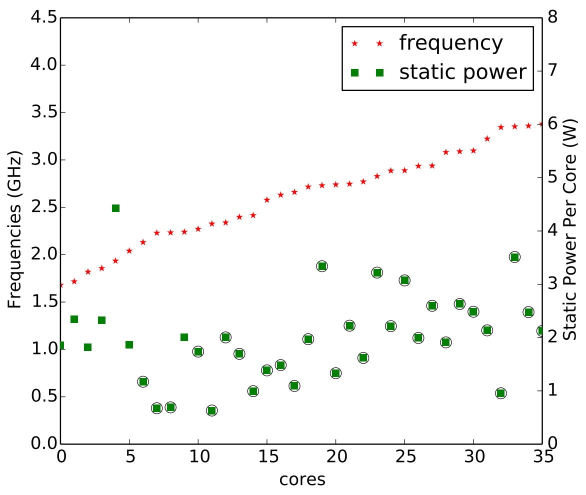
(a) MinFreq heuristic



(b) MaxFreq heuristic



(c) GoodCores heuristic



(d) ILP

Figure 4: An example of a configuration selected by the heuristics and the ILP optimization method for Jacobi application. Circle markers correspond to the cores selected by the corresponding method. MinFreq heuristic and ILP selected 29 cores each while MaxFreq heuristic and GoodCores heuristic selected 29 and 31 cores, respectively.

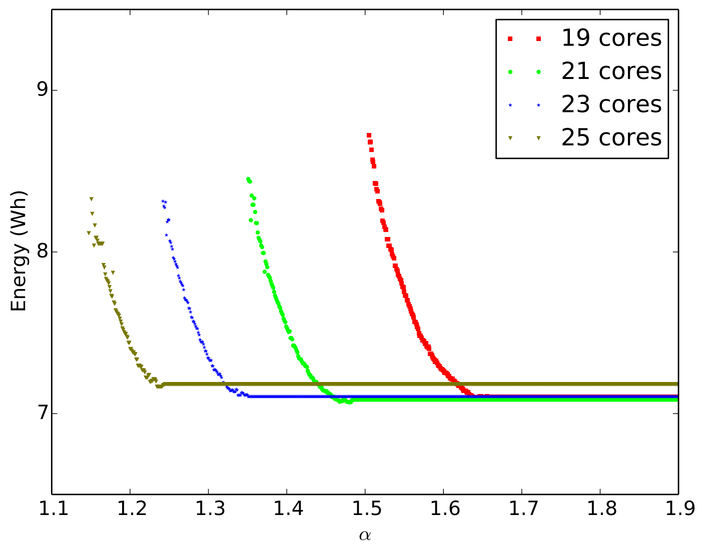


Figure 5: Energy value for 1000 values of α between 1 and 1.9 obtained by optimizing Program 4 of the ParSearch method for various values of K.

where *nsamples* is the number of samples. The integer linear program (Equation 17-20, Program 4) is then optimized for all these values of α, and the value of α that gives the minimum energy configuration is selected. Table 2 shows the accuracy of this method with different number of samples of α, $nsamples \in \{4, 8, 12, 16, 20\}$. We evaluate ParSearch method for different execution time penalties, $tp \in \{150\%, 125\%, 100\%, 75\%, 50\%, 25\%, 15\%, 5\%\}$. Each entry in the table gives the value of

$$\frac{\text{Energy}_{\text{OPTIMAL}}}{\text{Energy}_{\text{ParSearch}}} \times 100,$$

where, $\text{Energy}_{\text{OPTIMAL}}$ is the optimal energy value obtained from the TransformedQP method, and $\text{Energy}_{\text{ParSearch}}$ is the best energy value obtained from the ParSearch method. As the table shows, ParSearch is very accurate even for small number of samples of α. In summary, ParSearch method selects the minimum energy value obtained by optimizing the ILPs for different values of α and all the values of $K \in [1, n]$.

As the number of samples increase, the accuracy increases in general but it is not necessary that it will increase because the samples may be different for different values of *nsamples*. Given the very high accuracy of the method, especially when time penalties are small, this method is a very good candidate for finding near-optimal configurations. In the following section, we show that the time to find the solution with this method is significantly smaller than the TransformedQP method, and hence this method is the desirable method for finding energy-optimal configurations.

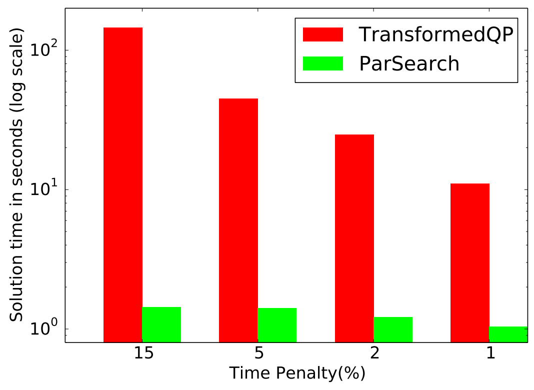
6.3. Solution Time

In this section, we compare the solution times for the TransformedQP method, the ParSearch method, and the exhaustive evaluation of all the configurations. For the experiments, we use a Dell 2.67 GHz Dual Westmere Xeon E5640 processor with a total of 8 cores and 16 SMT threads. The exhaustive evaluation of all the configurations for their energy consumption can be done in parallel. The total number of configurations to be evaluated for a 36 core chip is $2^{36} - 1$, which is equal to 68719476736 configurations. Exhaustive evaluation of all these configurations in parallel on 8 cores takes 74 hours, which makes this infeasible for online purposes.

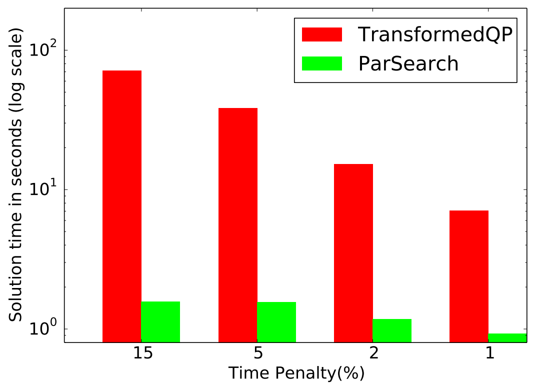
The TransformedQP method requires optimizing $n - 2$ proper ILPs, where n is the total number of cores on the chip. Each ILP has 702 variables, and 2000 constraints. The ILP optimizations are independent

Table 2: Accuracy of ParSearch for different execution time penalties and number of samples of α

Application	**Time Penalty (%)**	**Number of Samples**				
		4	**8**	**12**	**16**	**20**
Jacobi3d	**150**	99.72	99.77	99.81	99.81	99.82
	125	99.67	99.77	99.82	99.84	99.86
	100	99.76	99.83	99.82	99.86	99.88
	75	99.74	99.81	99.83	99.87	99.87
	50	99.7	99.73	99.84	99.88	99.9
	25	99.94	99.94	99.96	99.96	99.97
	15	99.82	99.87	99.91	99.94	99.93
	5	99.95	99.96	99.97	99.98	99.99
miniMD	**150**	99.73	99.79	99.75	99.8	99.79
	125	99.69	99.75	99.81	99.77	99.85
	100	99.54	99.55	99.73	99.77	99.84
	75	99.49	99.51	99.75	99.72	99.82
	50	99.63	99.68	99.78	99.87	99.91
	25	99.85	99.88	99.88	99.9	99.92
	15	99.92	99.94	99.96	99.97	99.97
	5	99.97	99.98	99.99	99.99	99.99



(a) Jacobi3d



(b) miniMD

Figure 6: Time to find the solution by TransformedQP method and ParSearch method with 20 samples of α

Table 3: Solution Time in seconds (average across 25 chips) for TransformedQP and ParSearch methods

Application	Time Penalty (%)	TransformedQP	ParSearch nsamples		
			4	12	20
Jacobi3d	15	146	0.41	0.89	1.44
	5	44.9	0.4	0.88	1.42
	2	24.8	0.37	0.77	1.22
	1	11.1	0.34	0.66	1.04
miniMD	15	71.4	0.46	0.98	1.57
	5	38.4	0.44	0.97	1.56
	2	15.26	0.38	0.79	1.18
	1	7.1	0.31	0.59	0.93

of each other and can therefore be very easily parallelized by launching them in parallel on multiple cores of a compute node and/or on multiple compute nodes. The ILP optimizations required for a given chip and an application were launched in parallel on the machine. The ParSearch method requires optimizing $(n-2) \times nsamples$ ILPs, where $nsamples$ is the number of samples of α. Each of the ILP in this method has 36 variables and just 2 constraints. As in the TransformedQP method, ILPs in the ParSearch method can also be optimized in parallel. Table 3 compares the total time to obtain the solution for the two methods.

We now consider the solution time for the TransformedQP and the ParSearch method with 20 samples of α, for each of the four cases presented in Section 6.1. These are also demonstrated in Figure 6.

1. When the execution time penalty is 15%, it took an average of 146, 1.44 seconds for obtaining the optimal result for Jacobi with TransformedQP, ParSearch method and 71.4, 1.57 seconds for miniMD, respectively.
2. When the maximum execution time penalty of 5% is enforced, the configuration search space for ILP optimization is reduced significantly as fewer configurations are feasible. It took an average of 44.9s, 1.42s to find the optimal solution for Jacobi, and 38.4, 1.56s for miniMD with TransformedQP, ParSearch method, respectively.
3. With the maximum execution time penalty of 2%, the search space is further reduced, and it took only 24.8s, 1.22s for Jacobi and 15.26s, 1.18s for miniMD to find the optimal solution using TransformedQP, ParSearch method, respectively.
4. Finally, with the maximum execution time penalty of just 1%, the search space is reduced to a very small number, and it took only 11.1s, 1.04s for Jacobi and 7.1s, 0.93s for miniMD to find the optimal solution using TransformedQP, ParSearch method, respectively.

Since HPC simulations run for several hours, the overhead of finding the optimal configuration is negligible as compared to the execution time of the jobs, which can be from hours to days. The results clearly show that the ParSearch method is significantly faster than the TransformedQP method, and hence is the method of choice as it also gives very accurate configurations (Section 6.2).

7. Conclusion and Future Work

Finding energy-optimal configurations for chips that have variation across cores is a hard problem because of billions of possible configurations that are available. We proposed a very fast GoodCores heuristic that gives significant energy savings when the allowed execution time penalty on the application is high. We then show how integer linear programming techniques can be used to obtain energy-optimal configurations for any execution time constraints with negligible overhead to obtain the solution. We proposed the ParSearch method to solve the constrained optimization problem that minimizes the energy consumption given any execution time constraint. We show that up to 13% savings in energy can be obtained with only 2% increase in the execution time. Whenever a job is scheduled for execution on selected chips, the ILP optimizer can be executed on the chip itself to determine the optimal configuration for the job, prior to actual execution of the job on that chip. In this way, no extra compute resources are required for optimal configuration selection.

This work shows the applicability of integer linear programming techniques to solve hard problems like this. There is a significant future work that follows from this work. The proposed methods can be evaluated for chips with very large number of cores. We also plan to improve the accuracy of the performance and power models used in this work, for chips with the large number of cores. Finally, we would like to evaluate the proposed methodologies for other HPC applications that are both computationally and memory intensive such as Adaptive Mesh Refinement [34], Lulesh [39], etc.

References

[1] Gurobi Optimization Inc. Software, 2014. `http://www.gurobi.com/`.

[2] IBM CPLEX Optimization Studio. Software, 2012. `http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/`.

[3] The Department of Energy Report on Top Ten Exascale Research Challenges. `http://science.energy.gov/~/media/ascr/ascac/pdf/meetings/20140210/Top10reportFEB14.pdf`.

[4] Advanced Micro Devices. BIOS and Kernel Developer's guide (BKDG) for AMD Family 15h Models 00h-0fh Processors. January 2012.

[5] T. Achterberg. SCIP: Solving Constraint Integer Programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[6] B. Acun, A. Gupta, N. Jain, A. Langer, H. Menon, E. Mikida, X. Ni, M. Robson, Y. Sun, E. Totoni, L. Wesolowski, and L. Kale. Parallel Programming with Migratable Objects: Charm++ in Practice. SC, 2014.

[7] A. Agarwal, S. K. Mathew, S. K. Hsu, M. A. Anders, H. Kaul, F. Sheikh, R. Ramanarayanan, S. Srinivasan, R. Krishnamurthy, and S. Borkar. A 320mV-to-1.2 V On-die Fine-grained Reconfigurable Fabric For DSP/media Accelerators in 32nm CMOS. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 328–329. IEEE, 2010.

[8] P. Alonso, M. F. Dolz, F. D. Igual, R. Mayo, and E. S. Quintana-Ortí. Dvfs-control techniques for dense linear algebra operations on multi-core processors. *Computer Science-Research and Development*, 27(4):289–298, 2012.

[9] P. Alonso, M. F. Dolz, R. Mayo, and E. S. Quintana-Ortí. Improving power efficiency of dense linear algebra algorithms on multi-core processors via slack control. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 463–470. IEEE, 2011.

[10] A. Banerjee, T. Mukherjee, G. Varsamopoulos, and S. Gupta. Cooling-aware and thermal-aware workload placement for green HPC data centers. In *2010 International Green Computing Conference*, pages 245–256, August 2010.

[11] C. Bash and G. Forman. Cool job allocation: measuring the power savings of placing jobs at cooling-efficient locations in the data center. In *Proceedings of the USENIX Annual Technical Conference*, pages 29:1–29:6, Berkeley, CA, USA, 2007. USENIX Association.

[12] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear programming and network flows.* John Wiley & Sons, 2011.

[13] B. Behle, N. Bofferding, M. Broyles, C. Eide, M. Floyd, C. Francois, A. Geissler, M. Hollinger, H.-Y. McCreary, C. Rath, et al. IBM Energyscale for POWER6 Processor-based Systems. *IBM White Paper*, 2009.

[14] R. D. Blumofe, C. F. Joerg, B. C. Kuszmaul, C. E. Leiserson, K. H. Randall, and Y. Zhou. Cilk: An efficient multithreaded runtime system. *Journal of Parallel and Distributed Computing*, 37(1):55–69, 1996.

[15] M. Broyles, C. Francois, A. Geissler, M. Hollinger, T. Rosedahl, G. Silva, J. Van Heuklon, and B. Veale. IBM Energyscale for POWER7 Processor-based Systems. *white paper, IBM*, 2010.

[16] T. E. Carlson, W. Heirman, and L. Eeckhout. Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 52. ACM, 2011.

[17] K. Choi, R. Soma, and M. Pedram. Dynamic Voltage and Frequency Scaling based on Workload Decomposition. In *Proceedings of the 2004 international symposium on Low power electronics and design*, pages 174–179. ACM, 2004.

[18] K. Choi, R. Soma, and M. Pedram. Fine-grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Tradeoff based on the Ratio of Off-chip Access to On-chip Computation Times. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(1):18–28, 2005.

[19] L. Dagum and R. Menon. OpenMP: an industry standard API for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[20] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Understanding the Future of Energy-performance Trade-off via DVFS in HPC Environments. *Journal of Parallel and Distributed Computing*, 72(4):579–590, 2012.

[21] J. Forrest. CBC (Coin-or Branch and Cut) Open-source Mixed Integer Programming Solver, 2012. *URL https://projects.coin-or.org/Cbc.*

[22] F. Glover and E. Woolsey. Converting the 0-1 Polynomial Programming Problem to a 0-1 Linear Program. *Operations Research*, 22(1):pp. 180–182, 1974.

[23] R. Graybill and R. Melhem. *Power aware computing.* Kluwer Academic Publishers, 2002.

[24] C.-H. Hsu and W.-c. Feng. Effective dynamic voltage scaling through cpu-boundedness detection. In *Power-Aware Computer Systems*, pages 135–149. Springer, 2005.

[25] S. Hsu, A. Agarwal, M. Anders, H. Kaul, S. Mathew, F. Sheikh, R. Krishnamurthy, and S. Borkar. A 2.8 GHz 128-entry$\times$152b 3-read/2-write Multi-precision Floating-point Register File and Shuffler in 32nm CMOS. In *VLSI Circuits (VLSIC), 2012 Symposium on*, pages 118–119. IEEE, 2012.

[26] C. Huang, G. Zheng, S. Kumar, and L. V. Kalé. Performance Evaluation of Adaptive MPI. In *Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming 2006*, March 2006.

[27] S. Huang and W. Feng. Energy-efficient cluster computing via accurate workload characterization. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 68–75, Washington, DC, USA, 2009. IEEE Computer Society.

[28] I. Kadayif, M. Kandemir, and U. Sezer. An integer linear programming based approach for parallelizing applications in on-chip multiprocessors. In *Proceedings of the 39th annual Design Automation Conference*, pages 703–706. ACM, 2002.

[29] U. R. Karpuzcu, K. B. Kolluru, N. S. Kim, and J. Torrellas. VARIUS-NTV: A microarchitectural model to capture the increased sensitivity of manycores to process variations at near-threshold voltages. In *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*, pages 1–11. IEEE, 2012.

[30] H. Kaul, M. Anders, S. Hsu, A. Agarwal, R. Krishnamurthy, and S. Borkar. Near-threshold Voltage (NTV) Design: Opportunities and Challenges. In *Proceedings of the 49th Annual Design Automation Conference*, pages 1153–1158. ACM, 2012.

[31] H. Kaul, M. Anders, S. Mathew, S. Hsu, A. Agarwal, R. Krishnamurthy, and S. Borkar. A 300mV 494GOPS/W Reconfigurable Dual-supply 4-Way SIMD Vector Processing Accelerator in 45nm CMOS. In *IEEE International Solid-State Circuits Conference, ISSCC 2009, Digest of Technical Papers, San Francisco, CA, USA, 8-12 February, 2009*, pages 260–261, 2009.

[32] H. Kaul, M. A. Anders, S. K. Mathew, S. K. Hsu, A. Agarwal, R. K. Krishnamurthy, and S. Borkar. A 320 mv 56 μw 411 gops/watt Ultra-low Voltage Motion Estimation Accelerator in 65 nm CMOS. *Solid-State Circuits, IEEE Journal of*, 44(1):107–114, 2009.

[33] R. Kumar, D. M. Tullsen, and N. P. Jouppi. Core architecture optimization for heterogeneous chip multiprocessors. In *Proceedings of the 15th international conference on Parallel architectures and compilation techniques*, pages 23–32. ACM, 2006.

[34] A. Langer, J. Lifflander, P. Miller, K.-C. Pan, , L. V. Kale, and P. Ricker. Scalable Algorithms for Distributed-Memory Adaptive Mesh Refinement. In *Proceedings of the 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2012)*, New York, USA, October 2012.

[35] A. Langer, E. Totoni, U. S. Palekar, and L. V. Kalé. Energy-efficient computing for hpc workloads on heterogeneous manycore chips. In *Proceedings of the Sixth International Workshop on Programming Models and Applications for Multicores and Manycores*, PMAM '15, pages 11–19, New York, NY, USA, 2015. ACM.

[36] R. Laundy, M. Perregaard, G. Tavares, H. Tipi, and A. Vazacopoulos. Solving Hard Mixed-integer Programming Problems with Xpress-MP: A MIPLIB 2003 Case Study. *INFORMS Journal on Computing*, 21(2):304–313, 2009.

[37] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 469–480. IEEE, 2009.

[38] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal. Adaptive, transparent CPU scaling algorithms leveraging inter-node MPI communication regions. *Parallel Computing*, 37(10-11):667–683, 2011.

[39] Lulesh. `http://computation.llnl.gov/casc/ShockHydro/`.

[40] A. Makhorin. The GNU Linear Programming Kit (GLPK). GNU Software Foundation, 2000. `http://www.gnu.org/software/glpk/glpk.html`.

[41] A. Merkel and F. Bellosa. Balancing power consumption in multiprocessor systems. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems*, EuroSys '06, pages 403–414, New York, NY, USA, 2006. ACM.

[42] N. B. Rizvandi, J. Taheri, and A. Y. Zomaya. Some Observations on Optimal Frequency Selection in DVFS-based Energy Consumption Minimization. *Journal of Parallel and Distributed Computing*, 71(8):1154–1164, 2011.

[43] B. Rountree, D. H. Ahn, B. R. de Supinski, D. K. Lowenthal, and M. Schulz. Beyond DVFS: A First Look at Performance Under a Hardware-enforced Power Bound. In *IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*, 2012.

[44] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz. Bounding Energy Consumption in Large-scale MPI Programs. In *Proceedings of the ACM/IEEE conference on Supercomputing*, pages 49:1–49:9, 2007.

[45] O. Sarood, A. Langer, A. Gupta, and L. V. Kale. Maximizing throughput of overprovisioned hpc data centers under a strict power budget. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '14, New York, NY, USA, 2014. ACM.

[46] O. Sarood, A. Langer, L. V. Kale, B. Rountree, and B. de Supinski. Optimizing Power Allocation to CPU and Memory Subsystems in Overprovisioned HPC Systems. In *Proceedings of IEEE Cluster 2013*, Indianapolis, IN, USA, September 2013.

[47] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg. Fast: Frequency-aware Static Timing Analysis. *ACM Transactions on Embedded Computing Systems (TECS)*, 5(1):200–224, 2006.

[48] R. Springer, D. K. Lowenthal, B. Rountree, and V. W. Freeh. Minimizing execution time in MPI programs on an energy-constrained, power-scalable cluster. In *Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '06, pages 230–238, New York, NY, USA, 2006. ACM.

[49] Q. Tang, S. Gupta, D. Stanzione, and P. Cayton. Thermal-aware task scheduling to minimize energy usage of blade server based datacenters. In *2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing*, 2006.

[50] Q. Tang, S. Gupta, and G. Varsamopoulos. Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach. *IEEE Transactions on Parallel and Distributed Systems*, (11):1458–1472, November 2008.

[51] E. Totoni. *Power and Energy Management of Modern Architectures in Adaptive HPC Runtime Systems.* PhD thesis, Dept. of Computer Science, University of Illinois, 2014.

[52] G. L. Valentini, W. Lassonde, S. U. Khan, N. Min-Allah, S. A. Madani, J. Li, L. Zhang, L. Wang, N. Ghani, J. Kolodziej, et al. An overview of energy efficiency techniques in cluster computing systems. *Cluster Computing*, 16(1):3–15, 2013.

[53] S. Venugopalan and O. Sinnen. Optimal linear programming solutions for multiprocessor scheduling with communication delays. In Y. Xiang, I. Stojmenovic, B. Apduhan, G. Wang, K. Nakano, and A. Zomaya, editors, *Algorithms and Architectures for Parallel Processing*, volume 7439 of *Lecture Notes in Computer Science*, pages 129–138. Springer Berlin Heidelberg, 2012.

[54] A. Vishnu, S. Song, A. Marquez, K. Barker, D. Kerbyson, K. Cameron, and P. Balaji. Designing energy efficient communication runtime systems: a view from pgas models. *The Journal of Supercomputing*, 63(3):691–709, 2013.

[55] S. Wallace, V. Vishwanath, S. Coghlan, Z. Lan, and M. E. Papka. Measuring Power Consumption on IBM Blue Gene/Q. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 853–859. IEEE, 2013.

[56] L. Wang, S. U. Khan, D. Chen, J. Kołodziej, R. Ranjan, C.-z. Xu, and A. Zomaya. Energy-aware Parallel Task Scheduling in a Cluster. *Future Generation Computer Systems*, 29(7):1661–1670, 2013.

[57] L. Wang, G. von Laszewski, J. Dayal, and T. Furlani. Thermal aware workload scheduling with backfilling for green data centers. In *Proceedings of the 2009 IEEE 28th International Performance Computing and Communications Conference (IPCCC)*, December 2009.

[58] L. Wang, G. von Laszewski, J. Dayal, X. He, A. Younge, and T. Furlani. Towards thermal aware workload scheduling in a data center. In *International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN)*, December 2009.

[59] G. Wen, J. Hong, C. Xu, P. Balaji, S. Feng, and P. Jiang. Energy-aware Hierarchical Scheduling of Applications in Large Scale Data Centers. In *Cloud and Service Computing (CSC), 2011 International Conference on*, pages 158–165. IEEE, 2011.