Run-time Performance and Power Optimization of Parallel Disparity Estimation on Many-Core Platforms

Charles Leech, University of Southampton, UK
Charan Kumar, IIT Hyderabad, India
Amit Acharyya, IIT Hyderabad, India
Sheng Yang, University of Southampton, UK
Geoff V. Merrett, University of Southampton, UK
Bashir M. Al-Hashimi, University of Southampton, UK

This paper investigates the use of many-core systems to execute the disparity estimation algorithm, used in stereo vision applications, as these systems can provide flexibility between performance scaling and power consumption. We present a learning-based run-time management approach which achieves a required performance threshold whilst minimizing power consumption through dynamic control of frequency and core allocation. Experimental results are obtained from a 61-core Intel Xeon Phi platform for the above investigation. The same performance can be achieved with an average reduction in power consumption of 27.8% and increased energy efficiency by 30.04% when compared to DVFS control alone without run-time management.

CCS Concepts: •**Theory of computation** → *Online learning algorithms;* •**Computing methodologies** → *Scene understanding;* •**Computer systems organization** → *Parallel architectures; Embedded systems;* •**Hardware** → *Power estimation and optimization;* •**Software and its engineering** → *Multithreading; Power management;*

ACM Reference Format:
Charles Leech, Charan Kumar, Amit Acharyya, Sheng Yang, Geoff V. Merrett and Bashir M. Al-Hashimi, 2017. Run-time Performance and Power Optimization of a Parallel Disparity Estimation Algorithm on Many-Core Platforms. *ACM Trans. Embedd. Comput. Syst.* V, N, Article A (YYYY), 20 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Stereo vision has become more pervasive in embedded and physically-constrained systems. Disparity estimation (DE) algorithms are used in stereo vision to calculate the depth of objects in a scene. They are used in such applications as video surveillance, autonomous vehicles and mobile robots [Cyganek and Siebert 2009]. Algorithms need to satisfy real-time performance demands, with high matching precision and low power consumption. The choice of estimation algorithm and implementation platform are both important factors to meet these constraints and produce a viable embedded stereo matching system.

With the appearance of stereo vision algorithms in dynamic environments, where constraints can change frequently, achieving run-time power scalability without sacrificing real-time performance has emerged as the next challenge in this domain. For example, in autonomous vehicles the performance requirement may be driven

This work was supported in parts by the EPSRC Grant EP/L000563/1 and the PRiME Programme Grant EP/K034448/1 (www.prime-project.org). Experimental data used in this paper can be found at https://doi.org/10.5258/SOTON/D0221

DOI: http://dx.doi.org/10.1145/0000000.0000000

by the content in the scene as objects closer require a higher frame-rate to update their depth faster [Paone et al. 2014]. We predict that software implementations on many-core architectures will allow dynamic scaling of algorithms and platforms to balance performance and power constraints. An adaptive run-time management approach has been developed which employs a regression-based learning method to find power/performance trade-offs between frequency and core scaling, with the aim of increasing energy efficiency and extending device battery life [Shafik et al. 2015]. To the best of our knowledge, this is the first study that investigates a many-core implementation of the DE algorithm, employing run-time management to achieve a trade-off between power and performance. This paper provides the following contributions:

— Evaluation of the performance and power characteristics of a parallel implementation of a leading DE algorithm within a many-core operating space.
— An adaptive run-time management approach for power and performance modeling and optimization of dynamic applications on many-core systems.
— Experimental validation of the run-time management approach for the DE algorithm using power and performance trade-offs on a many-core platform.

The rest of the paper is organized as follows: Section 2 discusses related work into disparity estimation and run-time management. Section 3 details the underlining algorithms for DE and their implementation. Section 4 describes profiling of the algorithm on a many-core platform. Section 5 introduces the adaptive run-time manager for optimizing the DE algorithm. Its validation and results follow in Section 6. Finally, Section 7 concludes the paper.

2. RELATED WORKS

2.1. Disparity Estimation

Stereo Vision for depth estimation and 3D sensing has been used across many embedded applications including person counting and tracking [Burbano et al. 2015], autonomous navigation and obstacle avoidance [Mendes and Wolf 2013; Oleynikova et al. 2015] and mobile robotics [Karakaya et al. 2014; Solak and Bolat 2015].

At the highest level, DE can be categorized into global and local algorithms. Global algorithms are formulated as an optimization across parts of the entire image. They produce precise results, with low average error rates in the calculation of disparity values [Scharstein and Szeliski 2002]. However, they typically have complex implementations with high memory and hardware demands which have the potential to limit scalability to higher resolution images. As a result, investigations have been made to implement dedicated hardware architectures of more precise algorithms, such as Semi Global Matching (SGM) [Gehrig et al. 2009; Banz et al. 2010] and Adaptive Support Weight (ADSW) [Ding et al. 2011; Perri et al. 2013].

In contrast, local stereo matching algorithms have reduced computational complexity and more localized memory requirements, relying on simpler aggregation strategies [Ttofis et al. 2016; L. Nalpantidis and Gasteratos 2008]. However, these algorithms are prone to disparity errors at depth discontinuity regions due to the use of a fixed local window shape and size [Yoon and Kweon 2006]. To improve matching accuracy, a few attempts have been made by combining or modifying existing algorithms and transforms [Ambrosch and Kubinger 2010; Baha and Larabi 2012; Zhang et al. 2009], the most recent Adaptive Support Weight (ADSW) methods are currently the most accurate [Gehrig et al. 2009; Ding et al. 2011; Perri et al. 2013]. They work by assigning different weights to the pixels in the support window based on their color or proximity to the central pixel. In this way, they aggregate only those pixels that lie at the same disparity, leading to improved quality at depth borders [Yoon and Kweon 2006].

Recently, the use of a Guided Image Filter (GIF) [He et al. 2013] in local ADSW algorithms has been proposed to reduce the complexity of cost aggregation, leading to a high-quality, fast and simple local DE algorithm [Hosni et al. 2011]. Due to the reduced complexity of this type of filter, the algorithm can operate at real-time framerates for HD images when implemented in a parallel structure [Ttofis et al. 2016]. This has resulted in the migration of software implementations entirely into the hardware domain on FPGAs [Gehrig et al. 2009; Banz et al. 2010; Ttofis et al. 2016].

We highlight the fact that these fixed hardware designs lack the ability to perform adaptations at run-time and that power-performance scalability is a key attribute for any application operating on an embedded system. We chose a local algorithm for our experimentation because it has scalability when implemented in software due to implicit parallelism and low data dependence properties. ADSW and GIF enhancements ensure a high quality disparity map in terms of bad pixel errors without sacrificing the algorithm's parallelism. Scalability enables operation across a range of power-performance points, depending on system constraints. Furthermore, the memory and computational resource requirements of embedded systems prevent the implementation of Global and SGM algorithms due to their irregular data access patterns and high complexity algorithms [Banz et al. 2010].

The approach considered in this work can be categorized as a passive stereo vision method, relying on correspondence in a stereoscopic image pair. Alternatively, active stereo vision approaches use a projector-camera setup with a single light source, such as infrared in the case of the Microsoft Kinect [Usachokcharoen et al. 2015; Stowers et al. 2011], or a structured light array [Park et al. 2015; Tahara et al. 2015] to perform depth estimation. The two approaches are not comparable as they operate on different data sources.

2.2. Run-time Power and Performance Optimization

Dynamic Voltage and Frequency Scaling (DVFS) and Dynamic Concurrency Throttling (DCT) are both run-time power optimization approaches that have received significant attention over recent years [Shafik et al. 2015; Cochran et al. 2011]. DVFS is used to reduce energy consumption by lowering the operating voltage and frequency whilst causing acceptable performance degradation [Etinski et al. 2012]. DCT selects the number of concurrent processing cores and threads during run-time to manage application parallelism and exchange performance for energy [Porterfield et al. 2013; Shafik et al. 2015]. Both DVFS and DCT control have been used in conjunction as run-time control approaches to achieve minimized energy consumption and a required performance target [Curtis-Maury et al. 2008; Hwang and Chung 2013]. These approaches are based on offline training to learn the system architecture followed by online performance prediction to guide run-time optimization and adaptation.

Existing energy minimization approaches for parallel applications have the following limitations. Firstly, existing approaches [Porterfield et al. 2013] and [Curtis-Maury et al. 2008] ignore energy minimization in the sequential part of the application, which can be significant. Secondly, these approaches [Curtis-Maury et al. 2008; Hwang and Chung 2013] use offline training processes to learn the system architecture and control DVFS and/or DCT. As a result, their models are limited to single use-cases and their scalability is poor for different many-core architectural allocations of the same application. To address these limitations, our approach encompasses the entire application execution period and uses scalable adaptation based on online model training and an iterative control process to achieve optimized frequency and core settings.

Stereo matching algorithms have been implemented as use-case applications for runtime resource management, with parameters exported that allow dynamic tuning of the trade-off between performance and result quality [Paone et al. 2014; Mariani

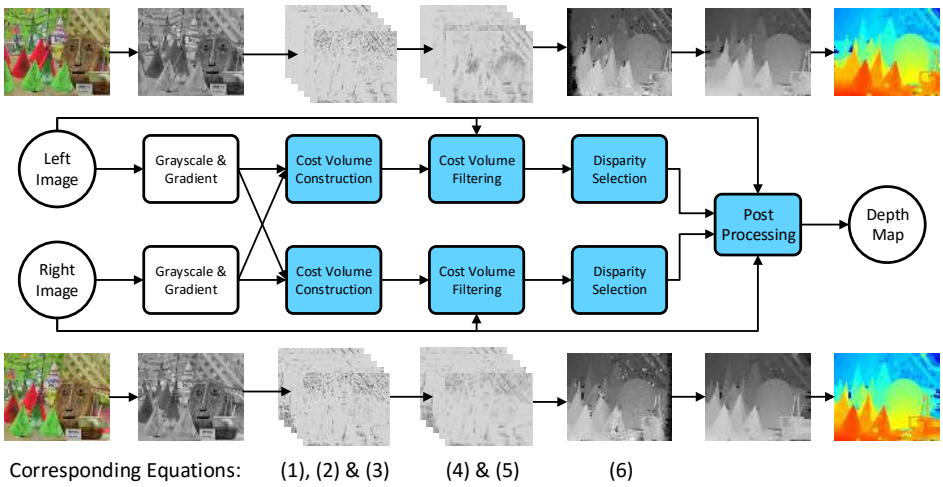


Fig. 1: Block diagram of the DE algorithm. The shaded regions are the enhanced parallel stages which offer opportunity for run-time adaptation.

et al. 2010]. To evaluate the runtime management approach, a dynamic workload is mimicked by randomly seeding start time, input data size and frame-rate parameters for multiple application instances. As in our work, comparison is often made to a baseline Linux configuration [Curtis-Maury et al. 2008] or a range of specific runtime manager (RTM) configurations [Paone et al. 2014].

3. DISPARITY ESTIMATION: ALGORITHM, IMPLEMENTATION AND VERIFICATION

Disparity estimation is a stereo matching process which can extract depth information from a pair of rectified, disparate images in a stereoscopic configuration. The correspondence of a pixel at coordinate (x, y) of the reference image, can be found at the same vertical coordinate y, within a maximum horizontal bound called the disparity range $[0, D)$ in the target image [Son et al. 2012]. The location difference of corresponding pixels in both images is called disparity and is used to calculate the depth in meters. DE algorithms mostly follow four high-level steps: cost computation, cost aggregation, disparity computation and disparity refinement, illustrated by Figure 1.

3.1. Algorithm

The algorithm is composed of four key stages. In addition, one stage of pre-processing is required to create an x-gradient version of the input image and via gray-scale conversion. Cost Volume Construction (CVC) is the comparison of pixels between the two images over a range of disparities d. A cost value is assigned to each pixel p in the left image based on the dissimilarity between it and a pixel in the right image. A truncated absolute difference of colors ($M(p, d)$, equation (1)) and gradients ($G(p, d)$, equation (2)) are the cost contributions, at pixel p and disparity d [Hosni et al. 2011]. I is the value of each pixel color channel in equation (1) and the luminance in equation (2). ∇_x is the gradient in the x direction of the luminance of the image.

$$M(p,d) = \sum_{i=1}^{3} |I^i_{left}(p) - I^i_{right}(p-d)| \tag{1}$$

$$G(p,d) = |\nabla_x(I_{left}(p)) - \nabla_x(I_{right}(p-d))| \tag{2}$$

A cost function (3) is used to balance the contribution from the color difference or gradient difference using a weighting variable α. T_c and T_g are bounding threshold values for the color and gradient cost contributions respectively for forming the overall cost. These variables are derived from the same literature as the algorithm and are set to the constant values: $\{\alpha, T_c, T_g\}$ = $\{0.9, 0.028, 0.008\}$[Hosni et al. 2011]. The parameters were found empirically by those authors who do not provide further information on how they were derived. They are the same for every frame.

$$C(p,d) = \alpha.min(T_c, M(p,d)) + (1-\alpha).min(T_g, G(p,d)) \tag{3}$$

Cost Volume Filtering (CVF), equation (4), is applied to each slice of the built cost volume. Filtering is performed using the Guided Image Filtering (GIF) method, using the original color image as the guidance image. $q(p,d)$ is the filtered cost value at pixel p and disparity d. A weighting function $W_{i,j}$ is used in the filter which favors pixels in the kernel that have similar color to that of the central pixel (see [He et al. 2013] for further details).

$$q(p,d) = \sum W_{i,j}(I)C(p,d) \tag{4}$$

Disparity selection (equation 5) reduces the cost volume down to a 2D disparity map through a winner-takes-all selection strategy. Selection finds the best disparity d_p value for each pixel p from the lowest cost value in the cost volume. D represents the upper bound of the disparity range ($0 \leq d < D$), within which the best disparity value must lie. The lowest cost represents the most likely distance of the same point in space between the two images. The disparity value not the cost value is encoded in the disparity map.

$$d_p = \underset{d \epsilon D}{\operatorname{argmin}}\, q(p,d) \tag{5}$$

Lastly, post processing is applied to the disparity map. A left-right consistency check, made possible because a disparity map is computed both from left to right and right to left, is used to identify and fill mismatched pixels between the two maps with the closest consistent pixel. A bilateral filter removes any remaining artifacts in the output disparity map by operating selectively only on the corrected pixel locations.

3.2. Parallelism

The four key stages of the algorithm, from CVC to Post Processing, are where parallelism has been introduced through multi-threading in order to create a number of independent threads. The algorithm is written in C++ with parallelism introduced using the POSIX threads library [Lewis and Berg 1998]. The parallel thread creation process is outlined in Algorithm 1, with the function called from $pthread\_create$ (line 12) containing the executed code. Lines 3 to 8 and line 19 describe how $levels$ thread blocks are created, each with $block\_size$ threads. Each thread in the block is allocated to a separate core therefore $block\_size$ determines the number of active cores. The first inner loop creates $block\_size$ threads, which execute simultaneously, then the second inner loop joins them before the next thread block is created. The $threads$ tuning parameter enables dynamic adaptation of the parallelism structure for each frame.

3.3. Verification and Accuracy

Correct algorithmic function must be observed in order to verify that a demanding workload is being presented to the system. In addition, functional correctness was

ALGORITHM 1: Thread creation process for each frame of the disparity estimation algorithm.

Input: number of threads: $threads$, disparity range: $maxDis = 64$

```
for CVC, CVF, Disparity Selection and Post Processing do
    level = 0
    repeat
        if level < maxDis/threads then
            block_size = threads;
        else
            block_size = maxDis (mod threads);
        end
        iter = 0
        repeat
            d = level * threads + iter;
            pthread_create(thread[d]);
        until iter = block_size;
    until levels = maxDis/threads;
end
```

Table I: Comparison of the accuracy of related stereo matching algorithms using standard image pairs from the Middlebury database.

Algorithm	Platform	Tsukuba			Venus			Teddy			Cones			% bad
		nonocc	all2	disc3	nonocc	all	disc	nonocc	all	disc	nonocc	all	disc	pixel
[Mei et al. 2011]	GPU	1.07	1.48	5.73	0.09	0.25	1.15	4.10	6.22	10.9	2.42	7.25	6.95	3.97
[Bleyer and Rhemann 2011]	CPU	2.09	2.33	9.31	0.21	0.39	2.62	2.99	8.16	9.62	2.47	7.80	7.11	4.59
[Wang et al. 2013]	FPGA	2.39	3.27	8.87	0.38	0.89	1.92	6.08	12.1	15.4	2.12	7.74	6.19	5.61
[Jin and Maruyama 2014]	FPGA	1.66	2.17	7.64	0.40	0.60	1.95	6.79	12.4	17.1	3.34	8.97	9.62	6.05
[Ttofis et al. 2016]	FPGA	2.38	3.01	9.38	0.40	0.7	3.62	7.23	12.7	17.2	2.87	8.59	8.27	6.36
Proposed	Xeon Phi	3	4.48	9.1	1.5	2.54	6.41	6	9.8	12.7	4.2	8.5	8.72	6.41
[Banz et al. 2010]	FPGA	4.1	-	-	2.7	-	-	11.4	-	-	8.4	-	-	6.7
[Hirschmuller 2008]	CPU	3.26	3.96	12.8	1.00	1.57	11.3	6.02	12.2	16.3	3.06	9.75	8.90	7.50
[Zhang et al. 2009]	CPU	1.99	2.65	6.77	0.62	0.96	3.20	9.75	15.1	18.2	6.28	12.7	12.9	7.60
[Shan et al. 2014]	FPGA	3.62	4.15	14.0	0.48	0.87	2.79	7.54	14.7	19.4	3.51	11.1	9.64	7.65
[Zhang et al. 2011]	FPGA	3.84	4.34	14.2	1.20	1.68	5.62	7.17	12.6	17.4	5.41	11.0	13.9	8.20
[Jin and Maruyama 2012]	FPGA	1.43	2.51	6.60	2.37	2.97	13.1	8.11	13.6	15.5	8.12	13.8	16.4	8.71

used to prove the successful introduction of parallelism and show that it does not adversely affect data accuracy. Table I shows that our algorithm is comparable, in terms of pixel errors per frame, to other works. Pixel error numbers are calculated for standard measures (nonocc, all, disc) across the four different image pairs of Tsukuba, Venus, Teddy and Cones, the later two of which are shown in Figure 2. The four input images (Figure 2a) and ground truth references (Figure 2c) were chosen from the widely-used Middlebury Stereo Vision dataset which provides a collection of stereo image resources for experimental purposes [Scharstein and Szeliski 2002]. The depth maps output from our algorithm are shown in Figure 2b for two test images for visual analysis of correct algorithmic function.

To analyse the effects of multi-threading and core scaling on the platform, experimentation with the algorithm described above is presented in the following section.

4. EXPERIMENTAL CHARACTERIZATION OF MANY-CORE PERFORMANCE AND POWER CONSUMPTION

State-of-the-art commercial embedded platforms do not feature high core counts, especially in architectures where cores are individually configurable, therefore in emulation of future embedded many-core systems, the 61-core Intel Xeon Phi coprocessor

(a) Left Input Image (b) Output Disparity Map (c) Ground Truth

Fig. 2: Comparison of the depth map output from our algorithm and the ground truth depth map provided with the dataset.

is used as a demonstrative platform for core scaling [Intel 2015]. Although it is not strictly an embedded platform from an energy efficiency perspective, it is characteristic of the many-core parallel architectures that may feature in future embedded platforms which is why it is used in this experimentation.

To highlight the importance of system optimization for highly (but not embarrassingly) parallel algorithms, Figure 3 shows the power and performance trade-offs for running the DE algorithm on the Xeon Phi. Performance is defined as the Frames-Per-Second (fps) computed by the algorithm. Each point shows the performance and power consumption when executing the algorithm at each core count and frequency, sweeping active core number in intervals of 4 and frequency from 619 MHz through nine intervals to 1238 MHz. The labels attached to points show the number of active cores used at those operating points. Figure 3 shows that there is a 46W range in power consumption and over 15x range in performance that is attainable by operating at different frequency or core allocation points. Above 32 cores, speed-up of the algorithm from core scaling decreases and absolute performance drops towards 60 cores. Limitations in the scalability of the algorithm must be considered as well as the memory and interconnect subsystem of the MIC architecture, which is in a ring-main configuration. The implementation of the algorithm means that the entire cost volume must be accessed from main memory and stored again between each of the four stages. When 60 cores attempt to access this data at once, the memory subsystem, rather than the number of cores, is the limiting factor. This problem may be alleviated by reprogramming the application. In lieu of this, a run-time modeling approach can be used to find the optimal operating point.

Four example operating points are shown in Table II. Going from p1 to p2 can achieved similar performance (0.287 to 0.327) but with a 22% reduction in power consumption. Similarly, moving from p3 to p4 can yield a 3.1 times improvement in performance at approximately the same power consumption. For this application, using the maximum number of cores at maximum frequency (60 cores at 1238 MHz) does not yield the highest performance, yet it does consume the highest power. Therefore, DVFS alone is not sufficient to optimize power and performance. The next section introduces,

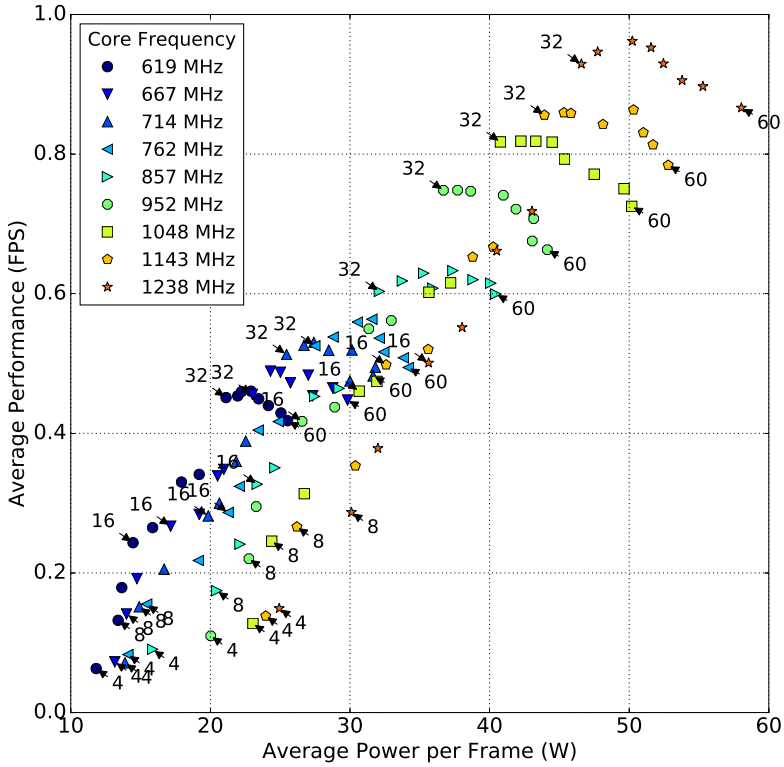


Fig. 3: Power and performance trade-offs for the possible range of cores and operating frequencies when executing the DE algorithm.

Table II: Normalized power and performance trade-offs.

Operating Point	p1	p2	p3	p4
Frequency (MHz)	1238	857	1238	667
Cores	8	16	4	32
Performance (fps)	0.287	0.327	0.149	0.456
Power (W)	30.10	23.35	24.93	23.01

adaptive run-time management which controls both frequency and core count to meet a target performance set by the DE algorithm.

5. PROPOSED ADAPTIVE RUN-TIME MANAGEMENT

Figure 4 shows the block diagram for the run-time optimization approach, highlighting the interactions between the application, run-time and hardware. The performance target of the application and power constraint of the system are communicated to the run-time layer through the application and system monitors framework. Adaptation of the core number and frequency is provided through the system and application controls framework. The run-time manager (RTM) consists of two components: a learning-based power/performance model and a run-time controller. These two components are explained in more detail in Sections 5.1 and 5.2.

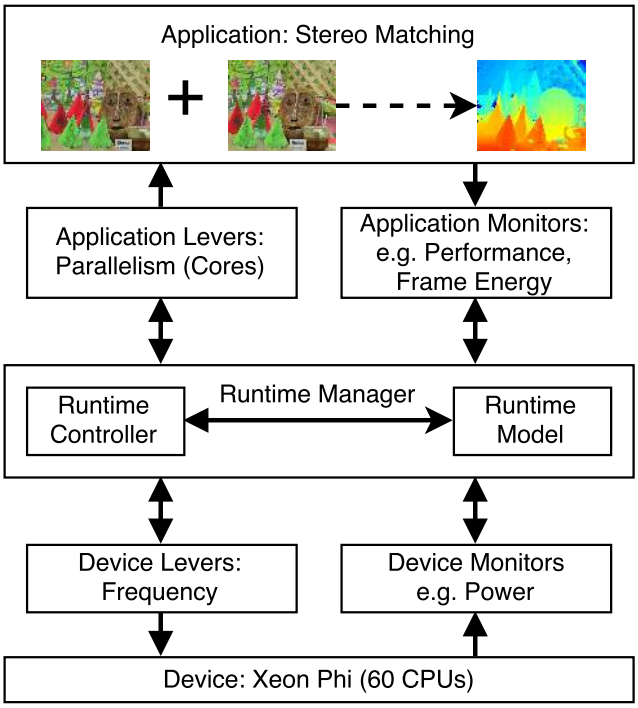


Fig. 4: Block diagram of the proposed run-time optimization approach.

5.1. Run-time Power and Performance Model

The run-time model emulates the current application and hardware configuration of the system (i.e. the DE algorithm on the Xeon Phi platform). Statically generated models can provide better accuracy and a more detailed operating space, but involve extensive offline profiling of individual applications, as in [Paone et al. 2014]. A run-time model enables flexibility in the application-system configuration as it can be relearned with a lower overhead than offline profiling, this benefit grows as the search space increases [Curtis-Maury et al. 2008]. In addition, a run-time model can be relearned if new applications begin executing or existing applications are updated. Through careful design of the run-time models, high accuracy can be achieved with low overhead (Section 5.2 and Section 6.2). Hence, our approach uses a run-time model as a critical component for energy-efficient adaptation [Yang et al. 2015]. Such a run-time model enables the prediction of power-performance trade-offs under different operating conditions. The model is learned using run-time measurements from power sensors and application performance measurements.

Figure 5 shows how the model is learned using linear regression in 4 steps. The modeling starts by varying the operating frequencies and number of active cores (step 1). For every frame, current and latency measurements are captured from the power sensors and the application (step 2). The measurements are used to test the hypotheses of the regression process until the learning interval is complete (Section 5.2 justifies the choice of the learning interval). After this interval, current and latency models are generated for the given application running on the platform (step 3). These models are combined to derive the power and performance models (step 4).

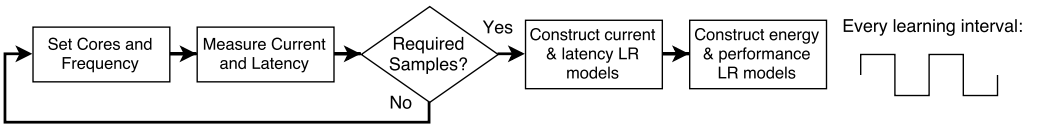


Fig. 5: Flowchart of the run-time energy/performance model generation.

Table III: Modeling Hypotheses.

Model	Hypothesis $h_\theta(x)$
Latency (τ)	$\theta_0 + \theta_1 \cdot \frac{1}{f} + \theta_2 \cdot \frac{1}{f \cdot c} + \theta_3 \cdot c$
Current (i)	$\theta_0 + \theta_1 \cdot v + \theta_2 \cdot v \cdot f \cdot c$
Performance	$F_{perf}(f) = 1/\tau$
Power	$F_{pow}(f) = v \cdot i$

Linear regression is used to establish a relationship between the dependent variables of the system (power and performance) and their associated independent predictor variables (i.e. number of cores, VF levels, etc.) [Cohen et al. 2013]. The relationship is defined by a hypothesis function as:

$$h_\theta(x) = \sum_{i=0}^{n} \theta_i x_i = \Theta^T X \tag{6}$$

where x_i is a predictor, n is the number of predictors, θ_i is a fitting coefficient, X and $\Theta^T X$ are the matrix representations of x_i and θ_i. The Θ values need to be chosen to minimize the mean-squared prediction error ($J(\theta)$) of the hypothesis in (6), which is given by:

$$J(\theta) = \sum_{j=1}^{m} (h_\theta(x^{(j)}) - y^{(j)})^2 = (\Theta^T X - \overrightarrow{y})^T (\Theta^T X - \overrightarrow{y}) \tag{7}$$

where y is the measured value, m is number of learning samples. $J(\theta)$ is minimum when its gradient becomes 0. Hence, from (7) the gradient of $J(\theta)$ can be defined as:

$$\nabla J(\theta) = \nabla(\Theta^T X - \overrightarrow{y})^T (\Theta^T X - \overrightarrow{y}) = X^T X \Theta - X^T \overrightarrow{y} \tag{8}$$

From (8), the fitting coefficients Θ of the hypothesis in (6) can then be computed as:

$$\Theta = (X^T X)^{-1} X^T \overrightarrow{y} \tag{9}$$

From (9), the computation complexity of the regression-based modeling is $O(n^2 \times m)$, where n is the number of predictors and m is the number of learning samples. Hence, to achieve a fast run-time model both n and m need to be small. In this work, two predictors are used: number of cores (c) and frequency (f), together with the intercept; hence, n=3.

Performance and power are not linear functions of frequency and the number of cores, therefore we first generate models for output current (i) and latency (τ), then build performance and power models from these. Table III shows the different hypotheses used to generate the models. Column 1 shows the target model and column 2 shows the hypothesis used. These models and their hypotheses are explained further as follows:

(1) Latency (τ) is expressed as a sum of four terms: first a constant (θ_0) delay contributed by factors independent of multi-threading and frequency (such as memory

contention, I/O setup, etc); the second term ($\theta_1 \cdot \frac{1}{f}$) is proportional to the CPU clock period representing the time spend by the sequential part of the application; the third term ($\theta_2 \cdot \frac{1}{f \cdot c}$) is proportional to both the clock period and number of cores, representing the time spent by the parallel part of the application; the last term ($\theta_3 \cdot c$) shows the latency related to the effects of multi-threading.

(2) Current (i) is expressed as a sum of three terms: $\theta_0 + \theta_1 \cdot v$ approximates the leakage current, while $\theta_2 \cdot v \cdot f \cdot c$ signifies the dynamic current.
(3) Performance in Frames-Per-Second (fps) is inversely proportional to latency.
(4) Power is a product of the instantaneous current (i) and supply voltage (v).

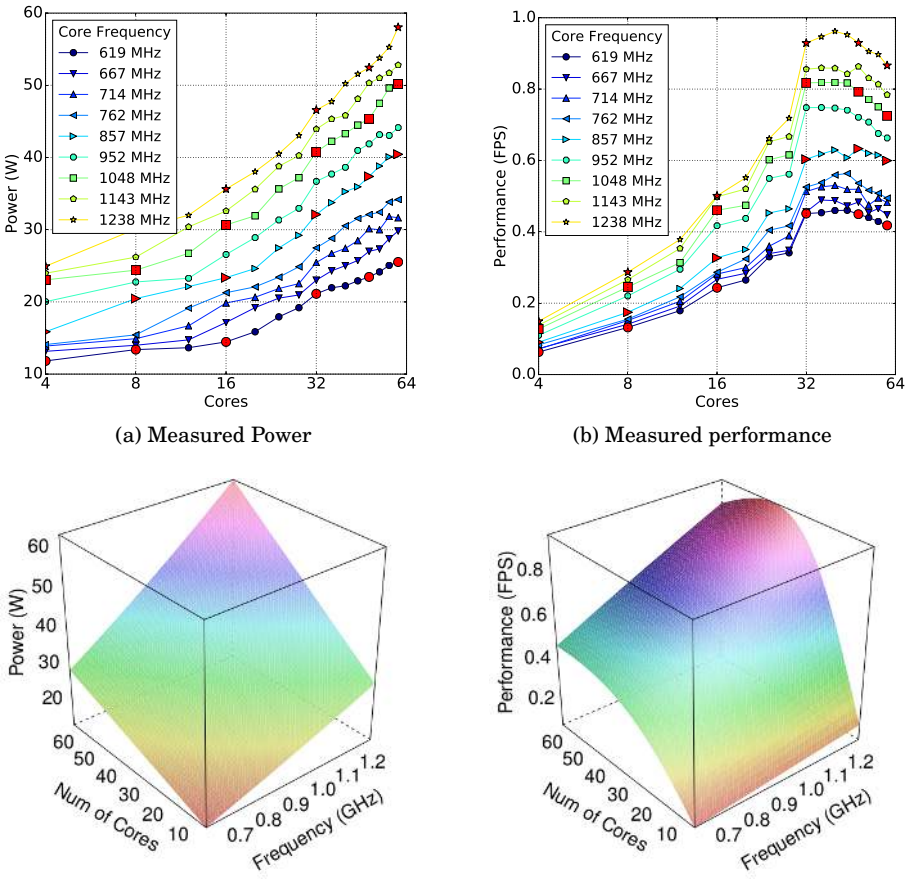
The supply voltage used in these hypotheses (Table III) is derived as a direct function of the operating frequency as it is fixed by the frequency controller based on the selected frequency. The regression-based learning of the power/performance trade-offs and their validations are further detailed in Section 5.2 where we demonstrate the impact that the number of learning samples has on the model prediction accuracy and associated run-time overheads.

5.2. Run-time Model Validation

Validation of the run-time model is carried out in two stages. In the first stage, the hypotheses of Table III are established as linear models. The models are then used at run-time in the second stage.

Figure 6a and 6b show plots of the measured power and performance of the DE algorithm executing on the platform. Testing across all different frequencies and number of cores would be necessary for a full design-space exploration. However, our run-time model has the advantage that only a subset of these operating points, highlighted in red, are required as training samples to generate the power and performance models of Figure 6c and 6d through the hypotheses of Table III. The models show predicted power and performance values across the full range of operating conditions. During application execution, the run-time models are used if the application or system constraints change to determine the new predicted optimal operating point.

The modeled values exhibit a high degree of correlation with the measured values. This is because the run-time model is generated using realistic component models of current (I) and latency (τ) from measured data. The accuracy of this model depends on a number of factors; the number of samples acquired, the number of predictors, and the underlying relationships between current, latency, performance and power. Figure 7 shows how many training points (frequency and core number) is required to achieve error convergence and low absolute power (Figure 7(a)) and performance (Figure 7(b)) modeling errors. The x-axis shows the number of core training points and the bars show the different number of frequency training points. The total number of training points is the product of the core and frequency training points. Modeling error is calculated using a comparison between the predicted power and performance values and measured data under the same conditions. The validation data set is exclusive of the training samples used to generate the models. The modeling error in terms of miss-prediction reduces with an increase in the number of training points. Convergence of the modeling error is significant after four core and three frequency training points. This is deemed as when the reduction in power and performance modeling error is less than 0.5% between training samples. The absolute performance and power model errors at this point are 5.95% and 4.25% respectively. This result shows that the run-time models require a low convergence interval of only 12 training points. This is because regression with a small number of predictors is rigid and the variance in the model is small [Draper and Smith 1998]. The high modeling accuracy helps the



(a) Measured Power

(b) Measured performance

(c) Power model

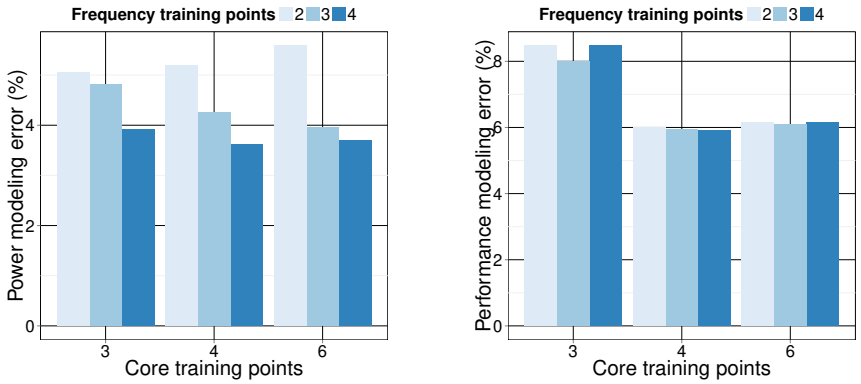
(d) Performance model

Fig. 6: Plots of measured power (a) and performance (b) with training data highlighted in red, and run-time models for power (c) and performance (d) generated from training data.

run-time manager to achieve near-optimal operational conditions. A demonstration of this is presented in Section 6.

5.3. Run-time Optimization

The run-time model enables optimization of power and performance at run-time through DVFS and core controls. Using the model coefficients, the run-time controller uses a gradient-descent based search in the optimization space to predict the number of cores and frequency given a power or performance constraint. Algorithm 2 shows example pseudo-code for the gradient-descent process when a performance target is used as the input. The objective is to find the minimum power point, frequency and core number. The operating frequencies (f_n) and number of cores (c_n) are initialized (line 1). These are updated by a gradient descent (line 3-4). The learning rate (α) in the algorithm is also initialized to a high value to ensure a fast convergence. While the updated F_{perf} exceeds the specified performance target (line 5), the learning rate is reduced (line 6) and f_n and c_n are updated by another gradient descent (line 7-8).



(a) Power modeling error

(b) Performance modeling error

Fig. 7: Evaluation of how increasing the number of training samples for both core count and frequency reduces the power and performance modeling error.

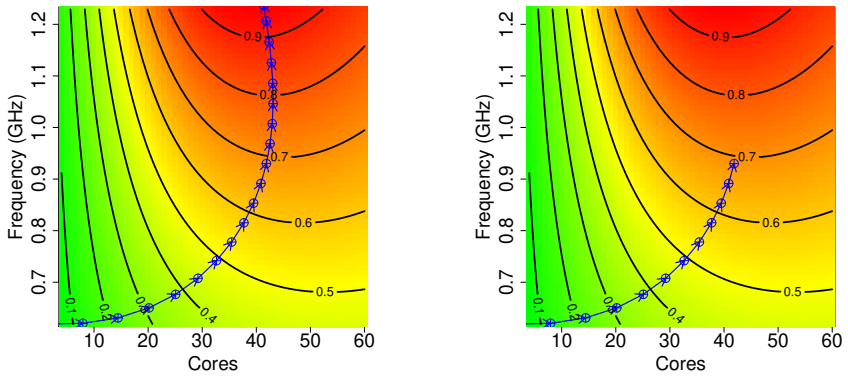
Predictions and updates are continued until the minimum performance target is met. The operating frequency and number of cores that provides the minimum power consumption while meeting the specified performance target are returned (f_n, c_n).

ALGORITHM 2: Gradient descent based performance constrained power optimization through DVFS and core control.

Input: Performance Requirement: $Perf_{req}$
Output: Minimum power point: Pwr_{min}, operating freq: f and number of cores: c

Initialize:f_n, c_n and learning rate: α
repeat
  $f_{n+1} := f_n - \alpha \frac{\partial}{\partial f} F_{perf}(f_n, c_n)$
  $c_{n+1} := c_n - \alpha \frac{\partial}{\partial c} F_{perf}(f_n, c_n)$
  while $F_{perf}(f_{n+1}, c_{n+1}) > Perf_{req}$ **do**
    $\alpha := 0.5 \times \alpha$
    $f_{n+1} := f_n - \alpha \frac{\partial}{\partial f} F_{perf}(f_n, c_n)$
    $c_{n+1} := c_n - \alpha \frac{\partial}{\partial c} F_{perf}(f_n, c_n)$
  end
  $f_{n-1} := f_n$, $f_n := f_{n+1}$
  $c_{n-1} := c_n$, $c_n := c_{n+1}$
until $f_n = f_{n+1}$ **&** $c_n = c_{n+1}$;
return $F_{perf}(f_n, c_n)$ *as* $Perf_{max}$

An example of DVFS and core controls through Algorithm 2 is shown in Figure 8a and illustrates how the gradient descent algorithm finds the optimum operating point. Power consumption is shown using an overlaid color map and the performance level is shown with contour lines. To determine the optimal DVFS controls and core allocation, the power and performance are predicted using the lowest operating frequencies (f_i) and core allocation (c_i) initially. For each next operating frequency during the search, the step size is reduced with the gradient. The gradient-descent heuristic search starts at one core with an operating frequency of $619MHz$, giving 0.017 predicted fps and $11W$, followed by 8 cores at $619MHz$ giving $0.12fps$ and $13W$. The process continues until it converges at 41 cores with a closest operating frequency of $1238MHz$ giving



(a) No Performance Constraint (b) Performance Constraint

Fig. 8: Run-time optimization examples (a) without performance constraint and (b) with performance constraint.

the highest performance of $0.99fps$ and an average power consumption of $57W$. Figure 8b demonstrates another example of the same algorithm applied with a performance target. This time the search only continues until the performance constraint is met, at which point the corresponding operating frequency and core allocation is selected (40 cores at $952MHz$). The effectiveness of run-time optimization of core number and DVFS controls is further validated in Section 6.

6. EXPERIMENTAL RESULTS - ADAPTIVE RUN-TIME MANAGEMENT

This section examines the effectiveness of the proposed approach, its overheads and verifies the accuracy of the DE algorithm's output. The proposed run-time approach is engineered into a prototype run-time manager and framework as part of a complete system; implementing application, run-time and hardware, on the Xeon Phi platform. The platform uses the same voltage-frequency island for all the cores. The operating frequency can be varied from $619MHz$ to $1238MHz$ in 9 steps, and the corresponding voltage varies from $0.995V$ to $1.060V$. The relationship between the supply voltage v and operation frequency f has been found empirically and can be approximated as $v(V) = 0.93 + 0.11f$ where frequency is in GHz. Core allocation and DVFS is controlled through the run-time manager using the run-time power/performance model.

6.1. Online Adaptation of the Disparity Estimation Algorithm

In section 4, the algorithm's power-performance operating space was characterized. This section demonstrates the online adaptation process that the RTM operates as part of a complete system. After run-time modeling of the algorithm's power and performance trade-offs is completed, the proposed approach can adapt to changes in the performance target through selection of the most energy-efficient core allocation and DVFS control settings. These properties are demonstrated in the three time series graphs of Figure 9. Furthermore, a comparison is made to the default Linux frequency governor and scheduler. These experiments are shown as dashed lines.

Changes in target performance are driven by external factors and depend on the particular application. In our case, a higher performance may be required to enable the system to calculate the depth of objects in the scene moving at a higher speed. Adaptation to changes in the performance target can be seen in the top series of Figure 9 where the measured performance tracks the target as it changes over time. The

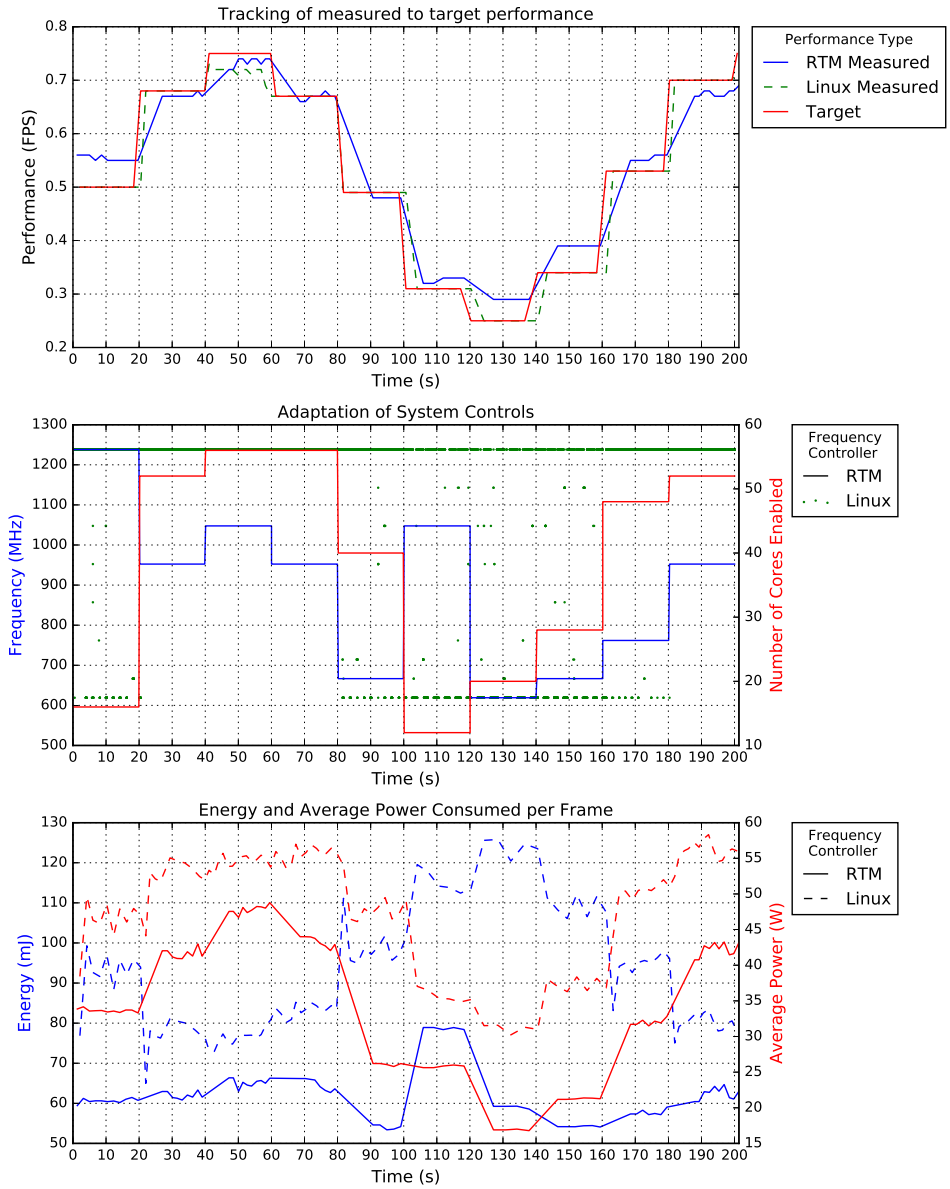


Fig. 9: Time series analysis of the RTM performing online adaptations of core number and frequency to optimize power and energy whilst meeting a target performance.

average absolute error in measured performance is 5.56%. Excluding occasions when the measured performance exceeds the target performance, which is not considered a penalty, this error drops to 1.16%. Performance of the application under Linux is manually matched to the target to allow the governor to make DVFS changes once frame processing is complete, as a result, the measured performance tracks the target.

The online adaptation of the number of active cores and their frequency is shown in the second series of Figure 9. The RTM uses the power/performance model to predict

Table IV: Comparison of the power and energy savings for the proposed approach to recently reported RTM approaches.

RTM	Modeling Method	Online/ Offline	DPM	% Power Saving	% Energy Saving
Porterfield et al. [2013]	Qthreads PMC	Online	DCT	-	3
Hwang and Chung [2013]	Heuristic	Online	DCT	18.12	-
Cochran et al. [2011]	Multinomial Logistic Regression	Online	DCT + DVFS	21	15.6
Curtis-Maury et al. [2008]	Multivariate Linear Regression	Offline	DCT + DVFS	6	19
Bellasi et al. [2015]	Profiling	Offline	DCT	-	25
Fu and Wang [2011]	Profiling + Feedback Control	Offline	DCT + DVFS	45	-
Proposed	Linear Regression	Online	DCT + DVFS	27.8	30.04

- no value reported

which operating point will give the target performance with the lowest power consumption. Fluctuations in the core number and frequency are due to the selection of overlapping of operating points, in terms of power and performance, between a high number of cores at a low frequency and a lower number of cores at a higher frequency. For example, at 112 seconds a target of 0.31 fps leads to 12 cores at 1048 MHz whereas at 154 seconds, the similar target of 0.34 selects 28 cores at 667 MHz. The section of operating points is non-trivial and so the RTM model is required to capture the power and performance trade-offs between DVFS and core scaling.

The final series shows the measured average power and energy consumed per frame as the performance target fluctuates, for both the proposed approach and the default Linux. To achieve the same performance, the proposed approach can reduce average power consumption by 27.8% and increase energy efficiency by 30.04%. The proposed approach consistently gives lower power consumption and increased energy efficiency across the entire range of performance targets. This is due to the core scaling ability of the RTM and its rigorous control of frequency.

Comparison to existing work, in terms of percentage power and energy saving, is detailed in Table IV. For each work, percentage savings were reported explicitly in the literature and relate to a comparison made between the proposed approach and a baseline/unmanaged configuration. The proposed approach gives higher power and energy savings compared to all other online approaches. Fu and Wang [2011] report higher percentage power savings, however this work relies on an extensive offline profiling component therefore it is not compatible with dynamic workloads. The short training phase for our approach enables the run-time model to be learned at the beginning of execution and relearned should the workload change. This property is discussed in more detail in the following section. Power and energy saving is not considered for works which perform run-time management on stereo matching algorithms [Gadioli et al. 2014; Mariani et al. 2010].

6.2. Run-time Manager Overheads

The proposed approach incurs run-time overheads due to the training phase and adaptation steps, including the optimization and control operations. As established in Section 5.2, collecting training data takes a period of 12 frames, which must be completed before optimization begins. Training time is dependent on the execution time of a single cycle of the application, for disparity estimation, it takes approximately 10 seconds depending on the exact operating points which are used as training frames. Learning the run-time model incurs an overhead of 2 ms, to calculate the linear regression coefficients. Run-time adaptation exhibits the average overhead of 2.4 ms due to the

Table V: Run-time manager training and adaptation/reconfiguration overheads.

RTM Approach	% Training Frames	% Total Adaptation Time
AS-RTRM	67.5	2.0
Proposed RTM	6	4.8

gradient-descent based search operation performed on the run-time model. Adaptation only takes place when the performance target or power constraint is changed.

For a 200 frame video sample, as used in [Paone et al. 2014], training with the proposed run-time approach has a 6% overhead in terms of the number of frames used. by comparison, an offline profiling approach has an overhead of 67.5% to map the complete operating space. This equates to an order of minutes in real time due to the need to sample all of the lowest performing configurations. Table V shows how our approach compares to a runtime resource management method where overheads are reported [Paone et al. 2014]. We present expected training times for our application and platform (to test all the required operating points). For adaptation overheads, we predict the behavior of our RTM operating in their workload scenario where six changes in performance requirement occur, therefore we would see an overhead of 14.4 ms (4.8%) over the 300s experiment time.

7. CONCLUSIONS

This work shows that run-time modeling is a critical component for the optimization of many-core systems executing parallel applications, enabling run-time adaptation and control of tuning parameters. In particular, this paper has analyzed the implementation of a parallel disparity estimation algorithm on the Intel Xeon Phi many-core platform to demonstrate a 46W range in power consumption and 15x range in performance, motivating the need for run-time management. An adaptive run-time power and performance optimization approach has been presented to reduce the power consumption of the system. We have reported a performance and power trade-off, demonstrating that it is possible to achieve the same performance with lower power consumption and higher energy efficiency by optimizing frequency and core allocation. In addition, our run-time based modeling approach has a low training overhead of only 12 frames to achieve error convergence and an online adaptation overhead of 4.8%.

REFERENCES

K. Ambrosch and W. Kubinger. 2010. Accurate Hardware-based Stereo Vision. *Comput. Vis. Image Underst.* 114, 11 (Nov 2010), 1303–1316. DOI:http://dx.doi.org/10.1016/j.cviu.2010.07.008

N. Baha and S. Larabi. 2012. Accurate Real-time Neural Disparity MAP Estimation with FPGA. *Pattern Recogn.* 45, 3 (March 2012), 1195–1204. DOI:http://dx.doi.org/10.1016/j.patcog.2011.08.005

C. Banz, S. Hesselbarth, H. Flatt, H. Blume, and P. Pirsch. 2010. Real-time stereo vision system using semi-global matching disparity estimation: Architecture and FPGA-implementation. In *Embedded Computer Systems (SAMOS), 2010 International Conference on*. 93–101. DOI:http://dx.doi.org/10.1109/ICSAMOS.2010.5642077

P. Bellasi, G. Massari, and W. Fornaciari. 2015. Effective Runtime Resource Management Using Linux Control Groups with the BarbequeRTRM Framework. *ACM Trans. Embed. Comput. Syst.* 14, 2, Article 39 (March 2015), 17 pages. DOI:http://dx.doi.org/10.1145/2658990

M. Bleyer and C Rhemann. 2011. PatchMatch Stereo - Stereo Matching with Slanted

Support Windows. In *British Machine Vision Conference 2011*. 1–11. http://publik.tuwien.ac.at/files/PubDat_201949.pdf

A. Burbano, S. Bouaziz, and M. Vasiliu. 2015. 3D-sensing Distributed Embedded System for People Tracking and Counting. In *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*. 470–475. DOI:http://dx.doi.org/10.1109/CSCI.2015.76

R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda. 2011. Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. 175–185. DOI:http://dx.doi.org/10.1145/2155620.2155641

J. Cohen, P. Cohen, S.G. West, and L.S. Aiken. 2013. *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences*. Taylor & Francis.

M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. de Supinski, and M. Schulz. 2008. Prediction Models for Multi-dimensional Power-performance Optimization on Many Cores. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*. 250–259. DOI:http://dx.doi.org/10.1145/1454115.1454151

B. Cyganek and J. P. Siebert. 2009. *Introduction to 3D Computer Vision Techniques and Algorithms*. Wiley-Blackwell.

J. Ding, J. Liu, W. Zhou, H. Yu, Y. Wang, and X. Gong. 2011. Real-time stereo vision system using adaptive weight cost aggregation approach. *EURASIP Journal on Image and Video Processing* 2011, 1 (2011), 1–19. DOI:http://dx.doi.org/10.1186/1687-5281-2011-20

N. R. Draper and H. Smith. 1998. *Applied Regression Analysis* (3rd ed.). Wiley-Blackwell.

M. Etinski, J. Corbalan, J. Labarta, and M. Valero. 2012. Understanding the Future of Energy-performance Trade-off via DVFS in HPC Environments. *J. Parallel Distrib. Comput.* 72, 4 (April 2012), 579–590. DOI:http://dx.doi.org/10.1016/j.jpdc.2012.01.006

X. Fu and X. Wang. 2011. Utilization-Controlled Task Consolidation for Power Optimization in Multi-core Real-Time Systems. In *2011 IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications*, Vol. 1. 73–82. DOI:http://dx.doi.org/10.1109/RTCSA.2011.65

D. Gadioli, S. Libutti, G. Massari, E. Paone, M. Scandale, P. Bellasi, G. Palermo, V. Zaccaria, G. Agosta, W. Fornaciari, and C. Silvano. 2014. OpenCL Application Autotuning and Run-Time Resource Management for Multi-core Platforms. In *2014 IEEE International Symposium on Parallel and Distributed Processing with Applications*. 127–133. DOI:http://dx.doi.org/10.1109/ISPA.2014.25

S. K. Gehrig, F. Eberli, and T. Meyer. 2009. *A Real-Time Low-Power Stereo Vision Engine Using Semi-Global Matching*. Springer Berlin Heidelberg, 134–143. DOI:http://dx.doi.org/10.1007/978-3-642-04667-4_14

K. He, J. Sun, and X. Tang. 2013. Guided Image Filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 6 (June 2013), 1397–1409. DOI:http://dx.doi.org/10.1109/TPAMI.2012.213

H. Hirschmuller. 2008. Stereo Processing by Semiglobal Matching and Mutual Information. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30, 2 (Feb 2008), 328–341. DOI:http://dx.doi.org/10.1109/TPAMI.2007.1166

A. Hosni, M. Bleyer, C. Rhemann, M. Gelautz, and C. Rother. 2011. REal-time local stereo matching using guided image filtering. In *2011 IEEE International Conference on Multimedia and Expo*. 1–6. DOI:http://dx.doi.org/10.1109/ICME.2011.6012131

Y. S. Hwang and K. S. Chung. 2013. Dynamic Power Management Technique for Multi-

core Based Embedded Mobile Devices. *IEEE Transactions on Industrial Informatics* 9, 3 (Aug 2013), 1601–1612. DOI:http://dx.doi.org/10.1109/TII.2012.2232299

Intel. 2015. Intel Xeon Phi Product Family. Online. (2015).

M. Jin and T. Maruyama. 2012. A Real-time Stereo Vision System Using a Tree-structured Dynamic Programming on FPGA. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. 21–24. DOI:http://dx.doi.org/10.1145/2145694.2145698

M. Jin and T. Maruyama. 2014. Fast and Accurate Stereo Vision System on FPGA. *ACM Trans. Reconfigurable Technol. Syst.* 7, 1, Article 3 (Feb. 2014), 24 pages. DOI:http://dx.doi.org/10.1145/2567659

S. Karakaya, G. Kkyildiz, C. Toprak, and H. Ocak. 2014. Development of a human tracking indoor mobile robot platform. In *Proceedings of the 16th International Conference on Mechatronics - Mechatronika 2014*. 683–687. DOI:http://dx.doi.org/10.1109/MECHATRONIKA.2014.7018343

G. C. Sirakoulis L. Nalpantidis and A. Gasteratos. 2008. Review of stereo vision algorithms: From software to hardware. *International Journal of Optomechatronics* 2, 4 (Jan 2008), 435–462.

B. Lewis and D. J. Berg. 1998. *Multithreaded Programming with Pthreads*. Prentice-Hall, Inc.

G. Mariani, C. Ykman-Couvreur, K. Zhang, L. Zhang, and G. Lafruit. 2010. An Efficient Run-Time Management Methodology for Stereo Matching Application. In *23th International Conference on Architecture of Computing Systems 2010*. 1–6. http://ieeexplore.ieee.org/document/5759019/

X. Mei, X. Sun, M. Zhou, S. Jiao, H. Wang, and Xiaopeng Zhang. 2011. On building an accurate stereo matching system on graphics hardware. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. 467–474. DOI:http://dx.doi.org/10.1109/ICCVW.2011.6130280

C. C. T. Mendes and D. F. Wolf. 2013. Real Time Autonomous Navigation and Obstacle Avoidance Using a Semi-global Stereo Method. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. 235–236. DOI:http://dx.doi.org/10.1145/2480362.2480413

H. Oleynikova, D. Honegger, and M. Pollefeys. 2015. Reactive avoidance using embedded stereo vision for MAV flight. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 50–56. DOI:http://dx.doi.org/10.1109/ICRA.2015.7138979

E. Paone, D. Gadioli, G. Palermo, V. Zaccaria, and C. Silvano. 2014. Evaluating orthogonality between application auto-tuning and run-time resource management for adaptive OpenCL applications. In *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*. 161–168. DOI:http://dx.doi.org/10.1109/ASAP.2014.6868651

M. G. Park, J. Park, Y. Shin, E. G. Lim, and K. J. Yoon. 2015. Stereo vision with image-guided structured-light pattern matching. *Electronics Letters* 51, 3 (2015), 238–239. DOI:http://dx.doi.org/10.1049/el.2014.3770

S. Perri, P. Corsonello, and G. Cocorullo. 2013. Adaptive Census Transform: A Novel Hardware-oriented Stereovision Algorithm. *Comput. Vis. Image Underst.* 117, 1 (2013), 29–41. DOI:http://dx.doi.org/10.1016/j.cviu.2012.10.003

A. K. Porterfield, S. L. Olivier, S. Bhalachandra, and J. F. Prins. 2013. Power Measurement and Concurrency Throttling for Energy Reduction in OpenMP Programs. In *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*. 884–891. DOI:http://dx.doi.org/10.1109/IPDPSW.2013.15

D. Scharstein and R. Szeliski. 2002. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *Int. J. Comput. Vision* 47, 1-3 (2002), 7–42.

DOI:http://dx.doi.org/10.1023/A:1014573219977
R. A. Shafik, A. Das, S. Yang, G. Merrett, and B. M. Al-Hashimi. 2015. Adaptive Energy Minimization of OpenMP Parallel Applications on Many-Core Systems. In *Proceedings of the 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures*. 19–24. DOI:http://dx.doi.org/10.1145/2701310.2701311
Y. Shan, Y. Hao, W. Wang, Y. Wang, X. Chen, H. Yang, and W. Luk. 2014. Hardware Acceleration for an Accurate Stereo Vision System Using Mini-Census Adaptive Support Region. *ACM Trans. Embed. Comput. Syst.* 13, 4s, Article 132 (2014), 24 pages. DOI:http://dx.doi.org/10.1145/2584659
S. Solak and E. D. Bolat. 2015. Distance estimation using stereo vision for indoor mobile robot applications. In *2015 9th International Conference on Electrical and Electronics Engineering (ELECO)*. 685–688. DOI:http://dx.doi.org/10.1109/ELECO.2015.7394442
H. Son, K. Bae, S. Ok, Y. Lee, and B. Moon. 2012. *A Rectification Hardware Architecture for an Adaptive Multiple-Baseline Stereo Vision System*. Springer Berlin Heidelberg, 147–155. DOI:http://dx.doi.org/10.1007/978-3-642-27192-2_19
J. Stowers, M. Hayes, and A. Bainbridge-Smith. 2011. Altitude control of a quadrotor helicopter using depth map from Microsoft Kinect sensor. In *2011 IEEE International Conference on Mechatronics*. 358–362. DOI:http://dx.doi.org/10.1109/ICMECH.2011.5971311
T. Tahara, R. Kawahara, S. Nobuhara, and T. Matsuyama. 2015. Interference-Free Epipole-Centered Structured Light Pattern for Mirror-Based Multi-view Active Stereo. In *2015 International Conference on 3D Vision*. 153–161. DOI:http://dx.doi.org/10.1109/3DV.2015.25
C. Ttofis, C. Kyrkou, and T. Theocharides. 2016. A Low-Cost Real-Time Embedded Stereo Vision System for Accurate Disparity Estimation Based on Guided Image Filtering. *IEEE Trans. Comput.* 65, 9 (2016), 2678–2693. DOI:http://dx.doi.org/10.1109/TC.2015.2506567
P. Usachokcharoen, Y. Washizawa, and K. Pasupa. 2015. Sign language recognition with microsoft Kinect's depth and colour sensors. In *2015 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*. 186–190. DOI:http://dx.doi.org/10.1109/ICSIPA.2015.7412187
W. Wang, J. Yan, N. Xu, Y. Wang, and F. H. Hsu. 2013. Real-time high-quality stereo vision system in FPGA. In *Field-Programmable Technology (FPT), 2013 International Conference on*. 358–361. DOI:http://dx.doi.org/10.1109/FPT.2013.6718387
S. Yang, R. A. Shafik, G. V. Merrett, E. Stott, J. M. Levine, J. Davis, and B. M. Al-Hashimi. 2015. Adaptive energy minimization of embedded heterogeneous systems using regression-based learning. In *2015 25th International Workshop on Power and Timing Modeling, Optimisation and Simulation (PATMOS)*. 103–110. DOI:http://dx.doi.org/10.1109/PATMOS.2015.7347594
K. Yoon and In S. Kweon. 2006. Adaptive support-weight approach for correspondence search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28, 4 (2006), 650–656. DOI:http://dx.doi.org/10.1109/TPAMI.2006.70
K. Zhang, J. Lu, and G. Lafruit. 2009. Cross-Based Local Stereo Matching Using Orthogonal Integral Images. *IEEE Transactions on Circuits and Systems for Video Technology* 19, 7 (2009), 1073–1079. DOI:http://dx.doi.org/10.1109/TCSVT.2009.2020478
L. Zhang, K. Zhang, T. S. Chang, G. Lafruit, G. K. Kuzmanov, and D. Verkest. 2011. Real-time High-definition Stereo Matching on FPGA. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. 55–64. DOI:http://dx.doi.org/10.1145/1950413.1950428