HetroCV: Auto-tuning Framework and Runtime for Image Processing and Computer Vision Applications on Heterogeneous Platform

Daihou Wang
Rutgers Discovery Informatics Institute
Rutgers University
Piscataway, USA
Email: daihou.wang@rutgers.edu

David J. Foran, Xin Qi
Rutgers Cancer Institute of New Jersey
New Brunswick, NJ USA
Email: foran@cinj.rutgers.edu
qixi@cinj.rutgers.edu

Manish Parashar
Rutgers Discovery Informatics Institute
Rutgers University
Piscataway, USA
Email: parashar@rutgers.edu

***Abstract*—With the wide adoption of high-performance processors and accelerators, large-scale computer vision applications have gained great performance improvement. However, it often requires extensive experiments and expertise to achieve optimal performance from manually-tuned programs, and the programs often need to be re-tuned when transplanted to a different platform, or using a different system configuration.**

To overcome this problem, in this paper we proposed HetroCV, a programmer-directed auto-tuning framework and runtime for computer vision applications on heterogeneous CPU-MIC platform. In HetroCV auto-tuning framework, computation units in the application pipeline are categorized in to one of three patterns: *Map*, *Stencil* and *MapReduce*, and program statistics are extracted from units' meta-information. Machine learning is adopted to train models for each pattern using the tuned parameters and program statistics from trial-run sets, so that when a new unit is presented, HetroCV auto-tuner can use the corresponding trained model to generate optimized tuning parameters. In HetroCV runtime, performance models for processor and co-processor are built to predict the prospective execution time of each computation unit in the application pipeline. We adopted the *maximum-throughput mapping strategy*, thus each unit would be mapped dynamically to the processor/co-processor queue, which would generate the minimum overall execution time.

Experiments on two medical image processing applications running on heterogeneous platform composed of Intel Xeon CPU and Intel Phi co-processor showed advanced performance over naive OpenMP tuning and Genetic Algorithm (GA) based heuristic tuning.

***Keywords*-Online auto-tuning; runtime; heterogeneous platform; Intel MIC architecture; image processing;**

I. Introduction

As the rapid development of digital imaging devices and the large increase of images and video generated from social media, there has been a surge of interest in image processing applications and softwares. At the same time, the image processing algorithms they depended on are evolving in both complexity and scale, which made them in great need of high performance computing implementations. Meanwhile, as the growing of high-performance field, there have been more and more types of high-performance computing devices developed and applied into HPC appications, from the early SMP devices, GPGPUs, to the more recent many-core architecture Intel Phi coprocessors [1]. To take advantage the computing power, extensive work have been devoted to accelerate the computational intensive image processing [2]–[5] and computer vision [6], [7] applications on high-performance platforms.

However, most of the work so far focused only on specific application and running platform, which made them unable to be directly ported to other platforms. The applications have to be re-coded or re-tuned on other platforms, which to a great extend stall the application and development of new algorithms. With high-heterogenity and high-diversity of platforms becoming the main stream computation power, there comes the great need for novel platform independent programming framework for image processing and computer vision applications. Hadile [8], [9] was proposed as a solution. The domain-specific language (DSL) was built upon PetaBricks [10], an heuristic auto-tuning framework. However, Hadile focused only on the stencil-centered computations, and did not provide optimization to other computations patterns existed in image processing pilelines. On the other hand, the high time cost of heuristic tuning (using PetaBricks framework) made it impractical for real-time image processing applications.

To overcome these shortcomings, in this paper, we proposed HetroCV, an auto-tuning framework and runtime for image processing applications on heterogeneous CPU-MIC platforms. HetroCV represents computation units in the image-processing pipelines with three basic patterns: Map, Stencil and MapReduce, and uses program statistics extracted from the computation units to predict the optimal tuning parameters on-line through machine learning. In addition, HetroCV build performance models for the processor and the coprocessor of the heterogeneous platforms, and map the tuned computation units to either processor queue or coprocessor queue using the *maximum throughput mapping strategy*. The main contributions of the paper are:

- Pattern-based representation for computation units in image processing pipeline.
- A machine learning based auto-tuning framework for

1530-2016/15 $31.00 © 2015 IEEE
DOI 10.1109/ICPPW.2015.21

multi-core processor and MIC co-processor.
- Runtime scheduler for dynamic work balance between processor and co-processor.

The rest of the paper is organized as follows. Pattern-based image-processing and optimization are introduced in Section II. HetroCV auto-tuning framework and runtime are detailed in Section III. Experiments on real image-processing pipeline and results are illustrated in Section IV. Section V discusses prior work on auto-tuning frameworks, followed by conclusion in Section VI.

II. Characterize Image-processing Computation

Recent works in computer vision and image processing field [19]–[21] have adopted the idea to consider computer vision and image processing applications as streams of computation units. We recognized that, based on computation and data access patterns, the streamed computation units can be grouped into several basic categories. And the computation units from the same category share the same parameter searching space for performance optimization. In HetroCV, we adopt the "streamed computation units" idea and categorized those computation units into 3 basic of computation patterns, and later use this pattern information as part of the meta-information in the auto-tuning of each computation unit.

A. Patterns in Image-processing Pipelines

Comparing to scientific computations which emphasis on matrix operation, image-processing applications show different computational pattern, most of which can be characterized by the following three patterns: *Map*, *Stencil*, and *MapReduce*.

Map-pattern. Map pattern is defined as a group of data-parallel operations which are independent from each other. This operation was usually operated on a single pixel or a line of pixels, such as the gray-value transformations or binary operations on an image. Though simple, map pattern contributes to a large amount of computation in most image-processing applications. Figure 1(a) illustrated one operation of Map-pattern.

Stencil-pattern. Stencil pattern is defined as a group of operations which was conducted on image patches, and the image patches have overlaps with each other. The filtering-like operations are typical stencil pattern operation. Though they can also be treated as naive data-parallel operations, efficient data access and reuse of the overlapped data create new challenges during parallel programming and optimization. Figure 1(b) showed one operation of stencil-pattern.

MapReduce-pattern. Here we adopt the computation flow introduced by Google [38], and define the MapReduce pattern similarly. MapReduce pattern is a 2-phase pattern, in which phase 1 is composed of group of independent, usually data-parallel computations; while phase 2 works on the combination of independent results from phase 1. The majority of clustering algorithm follows the map-reduce pattern. Figure 1(c) showed one operation of MapReduce pattern.

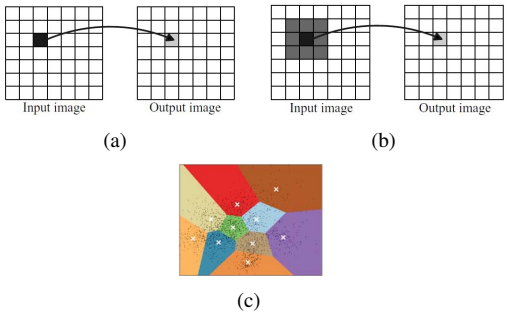


Figure 1. Illustrations of pattern-based image-processing operations. (a) Map-pattern operation, (b) Stencil-pattern operation, and (c) MapReduce-pattern operation.

B. Parallel Pattern-unit Computation

Extensive former studies in static tuning algorithm [8], [10], [25] have been working on code optimizations along two dimensions: computation granularity and storage granularity. So for performance optimization of pattern-based computations in HetroCV, we follow these two dimensions and extend them into the following sub-dimensions for computation units of each pattern.

Table I
Optimization parameters for pattern units

Pattern	Optimization Parameters
Map-pattern	$blockY\_num$, $blockX\_num$, $iSimd$, thN_i
Stencil-pattern	$blockY\_num$, $blockX\_num$, $iSimd$, $index\_PW$, $index\_BF$, $index\_SW$, thN_i
MapReduce-pattern	$blockY\_num$, $blockX\_num$, $iSimd$ iR_blockY_num, iR_blockX_num, thN_i

1) Tiling: Parallelizing the computation on an image by tiling is to distribute the data-independent computation onto parallel computation units. In HetroCV, we evenly distribute each image into $blockY\_num \times blockX\_num$ tiles.

2) Vectorization: To fully utilize the vector processing capabilities of the modern processors and co-processor [11]–[14]. In HetroCV, we use $iSimd$ to indicate whether the inner most loop of a set of nested loops was vectorized or not ($iSimd = 1$ indicates the inner most loop is vectorized, vice versa). s

3) Breadth first / sliding window: Breadth first and sliding window [8] are computation-granularity optimizations for stencil computations. In HetroCV, we adopt them as optimizations for Stencil-pattern computation units. For Stencil-pattern computations, we notice that, when the kernel size is relative small, neither breadth first nor sliding window can achieve better performance compared to original pixel-wise stencil computation. So in HetroCV, we use $index\_PW = 1$

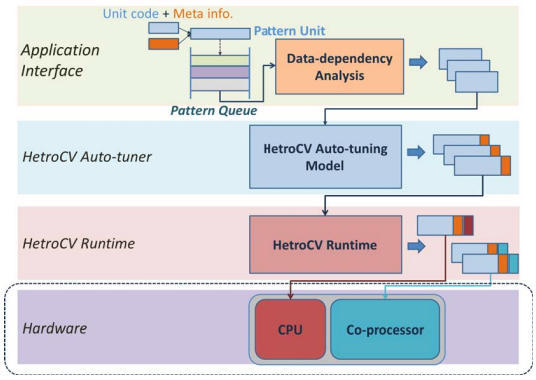


Figure 2. Overview of HetroCV auto-tuning framework and runtime.

to indicate pixel-wise stencil computation, in addition to $index\_BF = 1$ / $index\_SW = 1$ which indicates breadth first / sliding window stencil computation.

4) Partition: According to map-reduce programming model [38], the output of $map()$ functions would be partitioned into N sets as inputs to the N $reduce()$ functions. In HetroCV, for MapReduce-pattern, the output from $map()$ function were partitioned into $iR\_blockY\_num \times iR\_blockX\_num$ sets.

5) Multi-threading: In our CPU-MIC platfrom, both processor and MIC coprocessor support software multi-threading. So here use thN_P and thN_C to represent the OpenMP thread number used on the processor and co-processor, in which $thN_P \in \{8, 16, 32, 64\}$, $thN_C \in \{60, 120, 180, 240\}$.

In summary, the optimization parameters for computation units of each pattern are showed in Table I.

III. HetroCV Auto-tuning Framework and Runtime

In this section, we detailed the proposed HetroCV auto-tuning framework and rumtime for pattern-based image processing pipelines. Figure 2 gives an overview of HetroCV, which consists of 3 layers:

- Application interface. As a programmer directed tuning framework, in HetroCV, function variants and meta-information for computation units were gathered through the application interface. A pattern unit was generated to store the function variants and unit meta-information for each computation unit in the application. Pattern units were stored in a pattern queue. After an application was launched, the data-dependency analysis unit dequeues the "data-ready" pattern-unit for auto-tuning.
- HetroCV Auto-tuner. HetroCV auto-tuner built learning based models for optimization parameter predictions. The models were trained by a pre-defined training set at launch time. After that, optimization parameters were predicted for each dequeued pattern-unit.
- HetroCV Runtime. A performance model of the target hardware platform was built in HetroCV runtime. Predicted running time on processor and co-processor were generated, then the pattern-unit were mapped to the target processor/co-processor which gave the shortest overall task time.

A. Application Interface

1) Function variations and meta-information: In HetroCV, function variants were presented to the auto-tuning framework through application interface.

In addition to function variants, meta-information was passed through the application interface. Here in HetroCV, meta-information was composed of 2 parameters describing the computation unit it attached to. One parameter was an index indicating the pattern type of the unit ($iPattern\_type \in \{0, 1, 2\}$); the other parameter was a histogram describing the statistic of computation in the unit, $basicHist$ (detailed in section III-B2).

2) Data-dependency analysis: To achieve optimal performance, independent computation units should be executed in parallel. In HetroCV, a data dependency graph was built for all the patten units in the pattern queue, and updated after a pattern unit was added or completed. In this way, all the "data ready" pattern units will be tuned and launched in parallel.

B. HetroCV Auto-tuner

1) Learning based auto-tuning: Given an algorithm, auto-tuners work through the parameter space to search for the optimal parameter. However, high-dimension parameter often make exhaustively searching impractical for real-time applications. Even for the limited optimal parameter space we define in Tabel I, exhaustive search would not meet the performance requirement. So in HetroCV, we adopt the idea of the learning based auto-tuning.

We built 3 classifiers for the 3 types of computation units using support-vector machine [15]. Before the application was launched, exhaustive-searching based tuning was first performed on a set of training applications for each pattern. The classifiers were trained using the computation unit statistics and the optimal parameters from these sets. Later when a new pattern unit from the application arrives at the auto-tuner, the tuning classifier of this pattern type would predict optimal tuning parameters based on the computation unit statistic feature of the unit.

2) Computation unit statistic: Performance of each computation unit was determined by the computations it completes and their data access patterns, thus the optimal tuning parameters for this unit were also determined by the computations completed in the unit and the data access patterns. In HetroCV, we designed a histogram to describe the computation statistic feature of each computation unit.

For a computation unit CU_i, we define the computation statistic $basicHist_i$ as,

$$\begin{aligned} basicHist_i = [&num\_dRead_i, num\_dWrite_i, \\ &num\_Read_i, num\_Write_i, \\ &num\_addsub_i, num\_muldiv_i, \\ &num\_complex_i]. \end{aligned} \tag{1}$$

In which num_dRead_i and num_dWrite_i represent the numbers of *delayed-read* and *delayed-write* operations in computation unit CU_i. Here we define a delayed-read as a column-major order read, and a delayed-write as a column-major order write of an array. As in HetroCV, the images were stored in memory as a row-major array, a column-major order read/write to an image array would have lower access speed than a row-major order access. And num_Read_i, num_Write_i represent the number of row-major order read/write operations in computation unit CU_i.

num_addsub_i represents the number of addition or subtraction operations in computation unit CU_i; num_muldiv_i represents the number of multiplication or division operations in the computation unit, and $num\_complex_i$ represents other complex computations as square root operation in the computation unit.

Given the function variations for processor and coprocessor, the computation unit statics are calculated from the dynamic instructions generated by the Intel compiler [28]. The compiler generates intermediate assembler-level instructions from Intel AVX [29] and SSE [30] instruction sets. The total number of instructions per thread is based on the thread number and data partition on each threads. We use instructions from all threads to calculate the computation unit statics of each categories: memory access, computation and control.

3) Training Set: To train the auto-tuning classifier for each pattern type of computation units, we built a training set for each pattern from element image-processing operations, showed in Table II (in which type "M", "S", "MR" stand for Map-pattern, Stencil-pattern and MapReduce-pattern respectively). Those operations were selected as we try to use typical yet diverse element operations to represent computation from each pattern.

Table II
AUTO-TUNING CLASSIFIER TRAINING SET APPLICATIONS

Training Application	Pattern Type
RGB to HSI color-space Transformation	M
Gradient computation	M
Mean filtering	S
Gaussian filtering	S
Histogram Calculation	MR

RGB to HSI color-space transformation. Pixel-wise operation to transform the pixel values under RGB color model to the ones under HSI model.

Gradient computation. Column-wise and row-wise operation to compute gradient image Im_gx and Im_gy from gray-level image Im_gray.

Mean filtering. Separable image smooth operation by replacing each pixel value with the average value in a $(2r_{filter}+1) \times (2r_{filter}+1)$ window centered at the pixel, in which r_{filter} is the filtering window size.

Gaussian filtering. Un-separable image smooth operation by replacing each pixel value with the value of the Gaussian kernel convolution of a $(2r_{filter}+1) \times (2r_{filter}+1)$ window centered at the pixel, in which r_{filter} is the filtering window size.

Histogram calculation Operation to calculate a $3 * bin\_num$-bin color histogram of image Im_rgb.

C. HetroCV Runtime

Offloading is the most widely used mode for computing on heterogeneous platforms composed of CPU(s) and coprocessor(s). Before an application was deployed on such platform, the computation intensive and non-intensive sections were identified from either trial-runs or theatrical analysis. Then the computation intensive sections would be programmed into a kernel function that can run on coprocessor(s).

By using offloading-mode, applications will benefit from the speed-up(s) of the kernel function(s), however, in some cases, data-transfering overhead between CPU and coprocessor becomes too large and deteriorates the overall performance. And for a certain application, this balance often varies based on the input data scale, which made hard-coded application using offloading not robust to input scale changes. On the other side, even for cases when the kernel speed-ups weren't out-beat by the overheads, offloading often leaves the CPU(s) idle and wait on the results from the coprocessos(s), which would to a waste of the CPU computing power.

To provide more robust performance, and increase the utilization of the CPU(s) on heterogeneous nodes, in this section, we introduce the HetroCV runtime for the heterogeneous CPU-Xeon Phi coprocessor node. The runtime is in charge of scheduling the application computation units base on a novel 2-phase dynamic mapping scheme. During the 1st phase, the mapping scheme acted upon a heuristic processor performance model got from trial-runs; then in the 2nd phase, the real time performance parameters of the computing node were learned on-line. In this study, we use the simple processor-accelerator model on the CPU-MIC computing node in the 1st phase of the mapping scheme.

1) Performance model and performance prediction: We adopt the heterogeneous platform performance model proposed in work [16], [39], and model the computing time on processor(P)/co-processor(C) T as combination of data movement time T^m and computation time T^c.

$$T_i = T_i^m + T_i^c, i \in \{P, C\} \tag{2}$$

In which the data movement time T^m equals a latency t^s for the first unit of data, plus the transfer time $dN * t^w$ for the following dN units of data, in which t^w represents the unit transfer time. The computation time T^c equals the amount of work unit cN divided by the processing rate η.

$$T_i^m = t_i^s + t_i^w * dN_i, i \in \{P, C\} \tag{3}$$

$$T_i^c = cN_i/\eta_i \tag{4}$$

From comparing the performances characters between processors and coprocessors, we know that processors have fast memory access rate and relatively slow computation rate compared to coprocessors, while coprocessors have slow memory access rate and fast computation rate. Here we can assume that the data movement latency and unit data transfer time can be neglected on processors, thus we have $t_c^s = t_c^w = 0$. Together with equation 2 3 and 4, we can have the approximate computing time on processor and coprocessor as

$$T_p = cN_p/\eta_p \tag{5}$$

$$T_c = t_c^s + t_c^w * dN_c + cN_c/\eta_c \tag{6}$$

$$T_i = ep_0 + ep_1 * dN_c, i \in \{P, C\} \tag{7}$$

In HetroCV, we adopt the approximate performance model from equation 5 and 6, and use linear fitting to get the performance model parameter $1/\eta_p$, t_c^s, t_c^w and $1/\eta_c$. The execution time from the trial-runs in auto-tuner training phase were used to calculate the initial value for the parameters. For each trial-run, the data transfer amount dN were extracted directly from the kernel functions, and computation amount cN were calculated from the computation statistics (from section III-B2) of each computation unit.

Given the computation statistic $basicHist_i$ of a computation unit CU_i, we can have the approximate total computation amount $compN_i$ through

$$compN_i = basicHist_i \cdot compHist_i \tag{8}$$

in which, $compHist_i$ is 7×1 vector concatenated from the element computation amount for each collum in computation statics $basicHist_i$. In our study, we use $compHist_i = [1, 1, 2, 2, 1, 2, 3]'$.

2) Computation unit scheduling: HetroCV runtime keeps 2 queues for efficient patten-unit mapping, one for processor and the other for coprocessor. The computation units mapped to each queue will be executed and dequeue on a FIFO basis. In the HetroCV processing pipeline, after a computation unit CU_i got the optimal parameters from the auto-tuner, expected performances for this unit on processor $expTp_i$ and coprocessor $expTc_i$ can be calculated from the performance prediction based on equation 8.

To maximize the throughput of the pipeline (minimize the response time), in HetroCV runtime, we adopted the

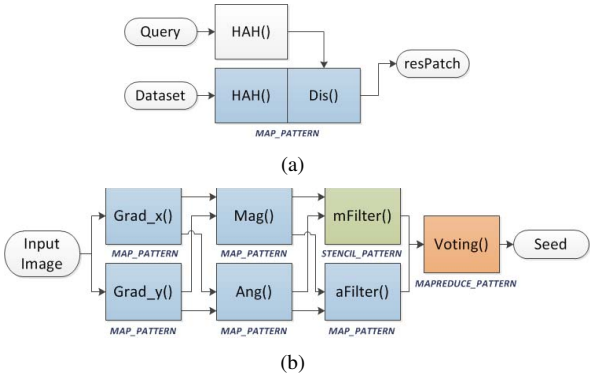


Figure 3. Workflow of the selected experiment applications. (a) Workflow of the histopathology cell retrieval application, (b) workflow of the histopathology cell detection application.

Maximum-throughput mapping strategy: a computation unit CU_i will be given a queue label $qL_i \in \{P, C\}$ and be mapped to processor queue ($qL_i = P$) or coprocessor queue ($qL_i = C$) whichever gives a smaller overall expected complete time for the whole queue $qT_i^+, i \in \{P, C\}$, with unit i added to the queue. The strategy can be illustrated as

$$qL_i = \begin{cases} \text{P} & \text{if } qT_p^+ < qT_c^+ \\ \text{C} & otherwise. \end{cases} \tag{9}$$

in which, the overall expected complete time equals the expected complete time of the queue and the expect performance time for unit i.

$$\begin{aligned} qT_p^+ &= qT_p + expTp_i, \text{if } qL_i = P \\ qT_c^+ &= qT_c + expTc_i, \text{if } qL_i = C \end{aligned} \tag{10}$$

$$\begin{aligned} qT_p &= \sum_u expTp_u, qL_u = P \\ qT_c &= \sum_v expTc_v, qL_v = C \end{aligned} \tag{11}$$

IV. Evaluation

To evaluate the performance of proposed HetroCV auto-tuning framework and runtime, we tested the HetroCV auto-tuner and runtime on 2 medical image processing applications. The applications are from our Computer-Aided Diagnosis(CAD) research [20] [21] at Center of Biomedical Imaging and Informatics, Rutgers RWJ Medical School. The workflow and the element pattern-units summary of the two applications are showed in Figure 3(a), 3(b) and Table III. Here, we use "CBIR" and "CD" as abbreviations for testing application **content-based image retrieval** and **histopathology cell detection**.

Content-based image retrieval Cell retrieval procedure uses color histogram as image feature, and works to retrieve image patches that shares resemblance to the input query image from the dataset images.

Table III
ELEMENT PATTERN-UNIT SUMMARY FROM TESTING APPLICATIONS

Pattern-unit Name	Pattern Type	Description
HAH+Dis (CBIR)	M	HAH [20] feature and distance calculation
Grad_x (CD)	M	Gradient calculation on x-direction
Grad_y (CD)	M	Gradient calculation on y-direction
Mag (CD)	M	Magnitude calculation of the 2D gradient vector
Ang (CD)	M	Phase angle calculation of the 2D gradient vector
mFilter (CD)	S	Gaussian filter on the 2d gradient vector magnitude
aFilter (CD)	M	Normalization filter on the 2d gradient phase angle
Voting (CD)	MR	Pixel-based voting algorithm [21]

Histopathology cell detection Cell detection procedure detects the center of all the cells in the image according to the gray-level information, and has been widely used as the first step of many histopathology analysis applications.

We use TACC Stampede [17] system as experimental testbed. Each compute nodes on Stampede is composed of 2 Xeon E5-2680 processors and 1 Intel Xeon Phi SE10P Coprocessor, connected through PCIe express interface.

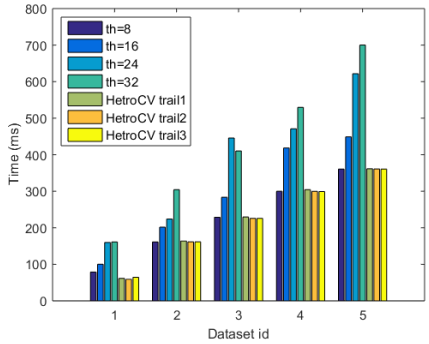
A. *HetroCV Auto-tuner*

1) HetroCV auto-tuner v.s *naive OpenMP tuning:* By using OpenMP, applications with data parallel computations would achieve performance increase on both CPU processor and Intel MIC co-processor; with the software threads number being the only tunable parameter. To give a thorough evaluation of the HetroCV auto-tuner, we first compared the pattern-units performances from HetroCV auto-tuner generated parameters with the ones got under different OpenMP software thread number. Here, we use thN_cpu and thN_mic to represent the OpenMP thread number used for testing on multi-core CPU and Xeon Phi co-processor respectively. In which, $thN\_cpu \in [8, 16, 24, 32]$, $thN\_mic \in [60, 120, 180, 240]$.

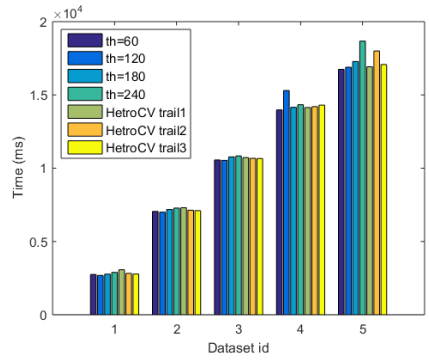
The performance comparisons of pattern-units from the **content-based image retrieval** application with multiple input data size are showed in Figure 4(a) for performances on multi-core CPU and Figure 4(b) for performances on Xeon Phi coprocessor. Performance comparisons of pattern-units from the **hitopathology image cell detection** application with variate input data size are showed in Figure 5 for performances on multi-core CPU and Figure 6 for performances on Xeon Phi coprocessor.

From the experiments we can see that, the optimal software thread number for OpenMP varied among different program units and among different input dataset sizes, which make it hard to achieve optimal performance through manually tuning. The results also proofed that, by adopting pattern-specific parameter space, HetroCV auto-tuner were able to out-perform the best performance that can be achieved from naive OpenMP tuning.

2) HetroCV auto-tuner v.s *heuristic searching:* Genetic algorithms [18] (GA) are a group of heuristic searching algorithms widely used for optimal parameter selection, and



(a)



(b)

Figure 4. Time performance comparison between tunable OpenMP thread number and HetroCV auto-tuner of computation sections from **content-based image retrieval** application on **multi-core CPU** and **Xeon Phi co-processor**.

have been adopted for parameter auto-tuning in various previous works such as PetaBricks [10] and Hadile [8]. To evaluate the performance of HetroCV auto-tuner, we compared the performance and tuning cost of HetroCV auto-tuner and the ones from GA-based tuning adopted in work [8] [10].

For **content-based image retrieval** application and **histopathology cell detection** application, the parameters listed in Table I composed a parameter space of 3-dimension and 24-dimension. Following work in [8], we built the initial population for the GA algorithm for faster convergence. The initial population was built from parameters indicating that

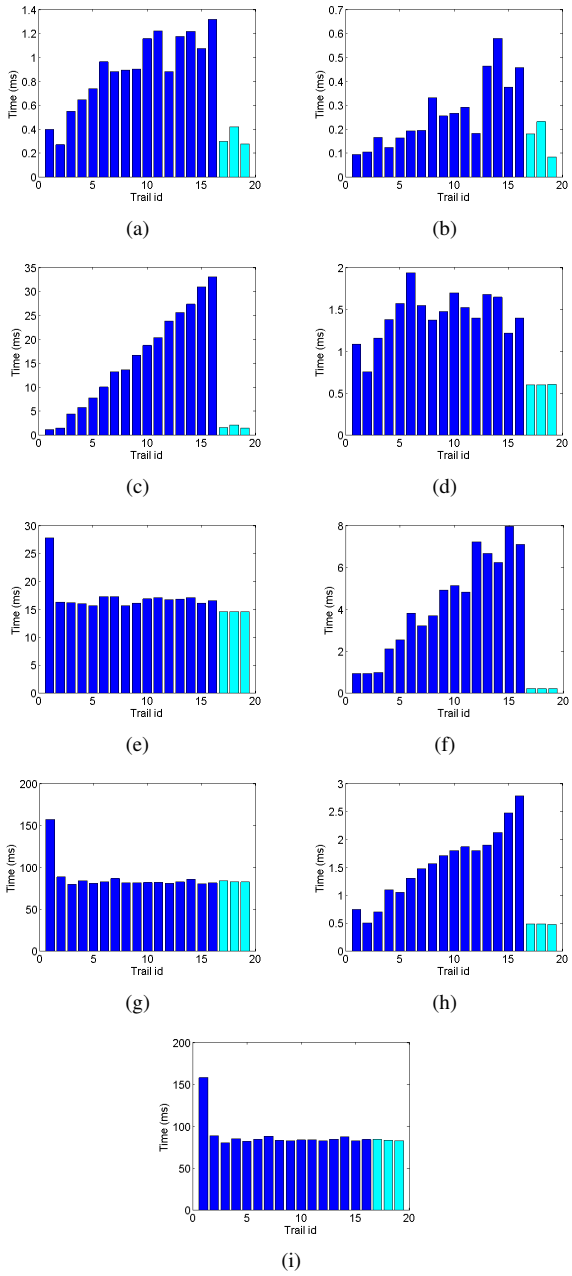


Figure 5. Time performance comparison between tunable OpenMP thread number and HetroCV auto-tuner of computation sections from **histopathology cell detection** application on **multi-core CPU**. In which, (a)(b)(c)(d)(e) show time performances of *Map-pattern* Grad_x(), Grad_y(), Mag(), Ang(), aFilter() section; (f) shows time performances of *Stencil-pattern* mFilter() section, and (g)(h)(i) show time performance of *MapReduce-pattern* Voting() section.

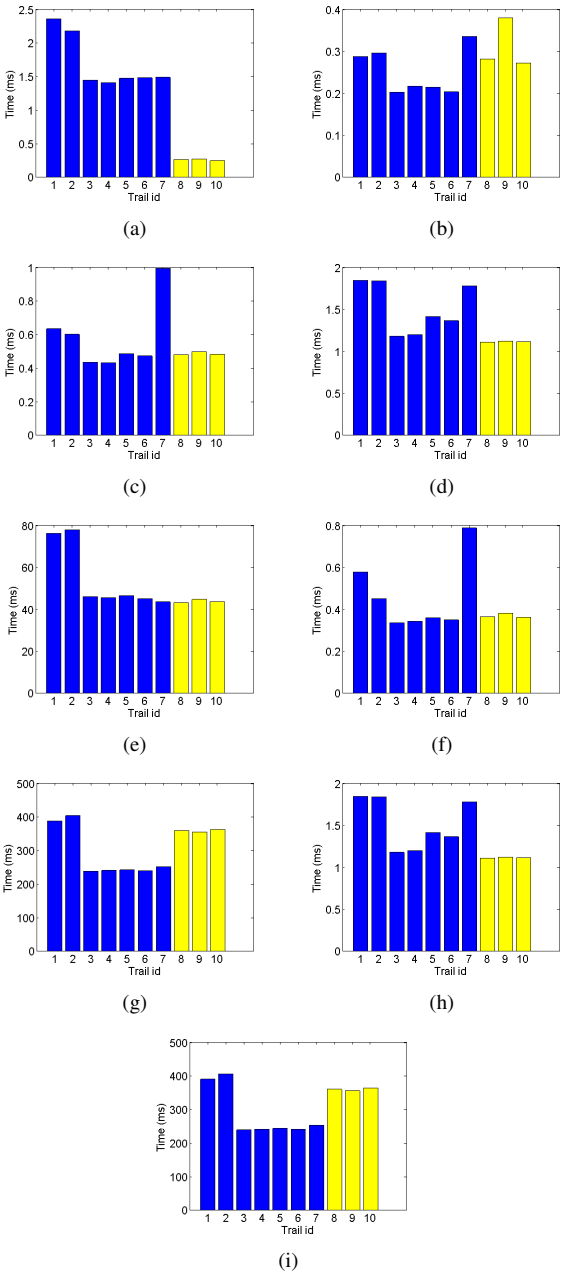


Figure 6. Time performance comparison between tunable OpenMP thread number and HetroCV auto-tuner of computation sections from **histopathology cell detection** application on **Xeon Phi co-processor**. In which, (a)(b)(c)(d)(e) show time performances of *Map-pattern* Grad_x(), Grad_y(), Mag(), Ang(), aFilter() section; (f) shows time performances of *Stencil-pattern* mFilter() section, and (g)(h)(i) show time performance of *MapReduce-pattern* Voting() section.

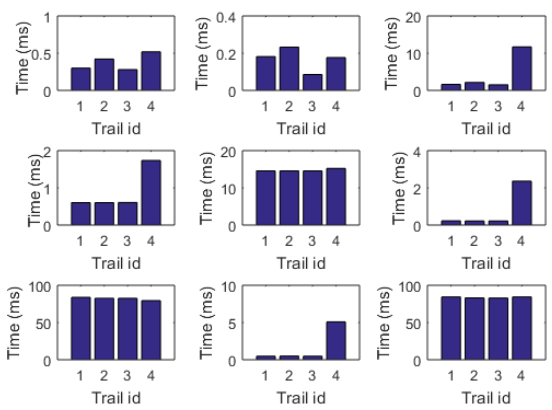


Figure 7. Performance comparison between HetroCV auto-tuner and GA-based tuning of computation sections from **histopathology cell detection** application. Trail $id = 1, 2, 3$ represent performance from HetroCV auto-tuning, trial $id = 4$ represent performance from GA-based tuning.

functions are deployed (1) *fully breadth first mode in both computation and storage*, or (2) *fully parallelized and tiled*, or (3) *parallelized over y-direction*. Then the population was expand with randomly generated populations.

In our experiment, we used 8-population/generation, and 50% elitism rate for GA iteration. The overall application performance was used as the convergence criteria. Figure 7 showed the performance comparison between HetroCV auto-tuner and GA-based tuning of computation sections from **histopathology cell detection** application.

B. HetroCV Runtime

The heuristic parameters in equation 7 and the unit data offload (uploading) time are summarized in Table IV.

V. Related Work

Large scale and data-intensive applications have been shown to benefit significantly from tuning and workflow optimization on both heterogeneous systems and multiple sites infrastructures [31]–[37]. Here we discuss some of the prior work.

Auto-tuning strategies were proposed to optimize the performance of applications with variable input scale and application parameters [22] [8]; under system load variation [23] or after transfered between platforms [25]. Automatic code generator and programmer directed auto-tuners have been proved to out-performed manually tuned applications on CPU-based [10] [23] and accelerator-based systems [24] [25].

Early auto-tuning strategies can be categorized into offline tuning and online tuning; learning-based and model-based tuning were later proposed. For offline tuning methods, a parameter space is built, and the optimal parameter is searched through heuristic searching over the parameter space. Genetic algorithms (GA) were adopted in PetaBricks [10] and Halide [8] for offline tuning. Although effective, offline tuning algorithm often result in over-length tuning time for larger parameter space.

Unlike offline tuning, online tuning methods chose optimal parameter at runtime. SiblingRavalry [23] used a two-path trial and online tuning to select the better performanced one at runtime. Online tuning methods are less computational expensive than offline methods, however it covers less of the parameter space.

Learning-based tuning methods were proposed to overcome the long searching time of offline tuning, by learning from the optimal parameters got from offline tuning of the training sets. Nitro [24] used statistic parameters of the input data as parameters, and trained a support-vector machine (SVM) for optimal parameter prediction. However, only certain statistic features from input dataset were adopted in Nitro. We extend this classification feature into both function and input data statistic parameters.

Auto-tuing can be more accurate by combing the platform information, J.Meng et al. [26] built an analytical model to tune the performance on CPU-GPU platform with the GPU performance model from work [27]. However, only limited research have been conducted on accurately modeling of the state of art platforms.

VI. Conclusion

In this paper we present HetroCV, a programmer-directed auto-tuning framework and runtime for computer vision applications on heterogeneous MIC platform. In HetroCV auto-tuning framework, the computational sections are categorized according to the computation and data access pattern, and the program statistics of the sections are provided as additional meta-information. Performance prediction models are built from trial run of training sets from each pattern, and when a new section is presented later, HetroCV can consult the corresponding model to select the appropriate tuning parameters. In HetroCV runtime, performance models for processor and co-processor are built to predict the prospective time performance for each application section, and the section would be mapped to processor/co-processor which have the shortest predicted computing time.

Experiments on 2 real-world medical image processing applications running heterogeneous platform composed of CPU-MIC(Intel Phi co-processor) computational nodes, showed promising performance comparing to manually tuning and Genetic Algorithm based tuning.

VII. Acknowledgments

This research was funded, in part, by grants from NIH contract 5R01CA156386-10 and NCI contract 5R01CA161375-03, NLM contracts 5R01LM009239-06 and 5R01LM011119-04, NSF Office of Industrial Innovation and Partnerships Industry/University Cooperative Research

Table IV
RUNTIME HEURISTIC PARAMETERS

Parameter type	Value
CPU computation time	$ep0 = 1.89, ep1 = 1.77e^{-8}$
MIC computation time	$ep0 = 5.69, ep1 = -2.63e^{-8}$
MIC data transfer time	$tTrans_{ON} = 8.54e^{-12}, tTrans_{OFF} = 6.57e^{-10}$

Center (I/UCRC) Program under award 0758596, and the National Center for Research Resources and the National Center for Advancing Translational Sciences, National Institutes of Health, through Grant UL1TR000117 (or TL1 TR000115 or KL2 TR000116). Additional support was provided by a gift from the IBM International Foundation. The project is also partially supported by the. The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIH.

Computing resources in this research is supported by the NSF Extreme Science and Engineering Discovery Environment (XSEDE), and Texas Advanced Computing Center (TACC).

REFERENCES

[1] Intel Many Integrated Core Architecture. https://software.intel.com/en-us/forums/intel-many-integrated-core

[2] M. Boyer, D. Tarjan, S. T. Acton, and K. Skadron, "Accelerating leukocyte tracking using CUDA: a case study in leveraging manycore coprocessors," in *Proceedings of IEEE International Symposium on Parallel & Distributed Processing (IPDPS 2009)*, pp.1-12

[3] G. Teodoro, T. Pan, T. M. Kurc, J. Kong, L. Cooper, N. Podhorszki, S. Klasky, and J. H. Saltz, "High-throughput analysis of large microscopy image datasets on CPU-GPU cluster platforms," in *Proceedings of IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS 2013)*, pp.103-114

[4] R. Membarth, F. Hannig, J. Teich, M. Krner and W. Eckert, "Generating device-specific GPU code for local operators in medical imaging," in *Proceedings of IEEE 26th International Symposium on Parallel & Distributed Processing (IPDPS 2012)*, pp.569-581

[5] J. P. Walters, V. Balu, S. Kompalli and V. Chaudhary, "Evaluating the use of GPUs in liver image segmentation and HMMER database searches," in *Proceedings of IEEE International Symposium on Parallel & Distributed Processing (IPDPS 2009)*, pp.1-12

[6] G. Poli, J. H. Saito, J. F. Mari and M. R. Zorzan, "Processing neocognitron of face recognition on high performance environment based on GPU with CUDA," in *Proceedings of 20th International Symposium on Architecture Computer Architecture and High Performance Computing (SBAC-PAD '08)*, pp.81-88

[7] C. Dwith and Rathna.G.N, "Parallel implementation of LBP based face recognition on GPU using OpenCL," in *Proceedings of 13th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2012)*, pp.755-760

[8] J. Ragan-Kelley, C. Barnes, A. Adams and et al, "Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '13)*, pp.519-530

[9] Halide. http://halide-lang.org/

[10] J. Ansel, C. Chan, Y. L. Wong, M. Olszewski, Q. Zhao, A. Edelman, and S. Amarasinghe, "PetaBricks: A language and compiler for algorithmic choice," in *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '09)*, pp.38-49

[11] Intel Xeon processor. http://www.intel.com/content/www/us/en/processors/xeon/xeon-processor-e5-family.html

[12] AMD. http://www.amd.com/en-us/products/processors

[13] Intel Phi coprocessor. http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html

[14] Nvidia Tesla. http://www.nvidia.com/object/tesla-supercomputing-solutions.html

[15] C. Hsu and C. Lin, "A comparison of methods for multiclass Support Vector Machines," in *IEEE Transaction of Neural Networks, Vol. 13, No. 2, March 2002*, pp.415-425

[16] D. W. Walker, "Performance at the exascale: a simple model for heterogeneous platforms", in *Proceeding of Clouds, and Data for Scientific Computing (CCDSC 2012).*

[17] Stampede. https://www.tacc.utexas.edu/stampede/

[18] M. Mitchell, *An introduction to Genetic Algorithms*. MIT Press, 1998.

[19] G. Teodoro, T. M. Kurc and et.al, "Accelerating large scale image analyses on parallel, CPU-GPU equipped systems," in *Proceeding of IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS 2012)*, pp.1093-1104

[20] X. Qi, D. Wang, I. Rodero, J. Diaz-Montes and et.al, "Content-based histopathology image retrieval using CometCloud," in *BMC Bioinformatics, 15:287, 2014*

[21] X. Qi, F. Xing, D. J. Foran, and L. Yang, "Robust segmentation of overlapping cells in histopathology specimens using parallel seed detection and repulsive level set," in *IEEE Transaction of Biomedical Engineering, Vol.59, No.3, 2012*, pp.754-765

[22] J. Shin, M. W. Hall and et al., "Speeding up Nek5000 with autotuning and specializaion," in *Proceedings of the 24th ACM International Conference on Supercomputing (ICS '10)*, pp.253-262

[23] J. Ansel, M. Pacula and et al., "SiblingRivalry: online autotuning through local competitions," in *Proceedings of the 2012 international conference on Compilers, architectures and synthesis for embedded systems (CASES '12)*, pp.91-100

[24] S. Muralidharan, M. Shantharam and et al., "Nitro: a framework for adaptive code variant tuning," in *Proceedings of the IEEE 28th International Parallel and Distributed Processing Symposium (IPDPS '14)*, pp.501-512

[25] P. M. Phothilimthana, J. Ansel and et al., "Portable performance on heterogeneous architectures," in *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems (ASPLOS '13)*, pp.431-444

[26] J. Meng, V. A. Morozov and et al., "GROPHECY: GPU performance projection from CPU code skeletons," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2011)*, pp.1-11

[27] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," in *Proceedings of the 36th annual international symposium on Computer architecture (ISCA '09)*, pp.152-163

[28] Intel C and C++ Compiler. https://software.intel.com/en-us/c-compilers

[29] Intel AVX. https://software.intel.com/en-us/blogs/2013/avx-512-instructions

[30] SSE. http://developer.amd.com/sse5.jsp

[31] P. Basu, M. Hall, M. Khan and et al, "Towards making autotuning mainstream," in *Int. J. High Perform. Comput. November 2013*, pp.379-393

[32] Z. DeVito, N. Joubert, F. Palacios and et al, "Liszt: a domain specific language for building portable mesh-based PDE solvers," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*, 2011.

[33] K. J. Brown, A. K. Sujeeth, H. J. Lee and et al, "A Heterogeneous Parallel Framework for Domain-Specific Languages," in *Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques (PACT '11)*, pp.89-100

[34] Delite. http://stanford-ppl.github.io/Delite/index.html

[35] A. K. Sujeeth , H. Lee , K. J. Brown and et al, "OptiML: an Implicitly Parallel Domain-Specific Language for Machine Learning," in *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*.

[36] OptiML. http://stanford-ppl.github.io/Delite/optiml/

[37] A. K. Sujeeth, A. Gibbons, K. J. Brown and et al., "Forge: generating a high performance DSL implementation from a declarative specification," in *Proceeding of 12th International Conference on Generative Programming: Concepts and Experiences (GPCE 13')*, October 2013.

[38] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Proceeding of Sixth Symposium on Operating System Design and Implementation (OSDI'04)*, San Francisco, CA, December, 2004.

[39] M. Boyer, J. Meng, K. Kumaran, "Improving GPU performance prediction with data transfer modeling," in *Proceeding of 27th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW 2013)*, pp.1097-1106