Mapping for a Heterogeneous Multi-Core Media Processor
Considering the Data Transfer Time

H. M. Waidyasooriya, D. Okumura, M. Hariyama and M. Kameyama
Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi, 980-8579, Japan
Email: {hasitha@kameyama., okumura@kameyama., hariyama@, kameyama@}ecei.tohoku.ac.jp

Abstract—Heterogeneous multi-core processors are attracted by the media processing applications due to their capability of drawing strengths of different cores to improve the overall performance. However, the data transfer bottlenecks and limitations in the task allocation have prevented us from gaining full potential of the heterogeneous multi-core processors. This paper presents a task allocation method to reduce the data transfer time and to increase the overall processing speed. According to the experimental results, the proposed method reduces 75% of the processing time compared to the conventional task allocation method.

Keywords: Heterogeneous, multi-core, task-allocation, system-on-chip

1. Introduction

Heterogeneous multi-core processors provide a platform to implement complicated media processing applications power efficiently. As shown in Fig.1(a), heterogeneous multi-core processors have different processor cores integrated on the same chip. If the tasks of an application are correctly allocated to the most suitable processor cores, all the cores work together to increase the overall performances. Examples of heterogeneous multi-core processors are [1] and [2]. The former has multiple cores of CPUs and ALU arrays. The latter has multiple cores of CPUs, a micro-controller and SIMD (single-instruction multiple-data) processors.

Mapping media processing applications on to a heterogeneous multi-core processor is a difficult task. Usually, the data-parallel tasks are mapped to the accelerator cores and the rest of the tasks are mapped to the CPU cores in a heterogeneous multi-core processor. Even though this increases the processing speed of individual tasks separately, it does not always increase the overall processing speed. We found two reasons that prevent the processing speed to be increased in heterogeneous processors. The first reason is the data transfer time between multiple cores. As shown in Fig.1(b), the heterogeneous multi-core processors have a hierarchical memory architecture. It contains a large memory (global memory) placed outside the accelerator core and several small memories (local memories) placed inside the accelerator core. Since the results of one task can be the inputs of another task allocated to a different core, we have to transfer data from the global memory to the local memories and from the local memories to the global memory, before and after the execution of the tasks respectively. This data transfer time is a huge problem especially in media processing, since they usually process a huge amount of pixel data. The second reason for the decrease in processing speed is the inability to perform calculations such as trigonometric operations, square-root and divisions in the accelerator cores. We call them accelerator-incompatible calculations. Therefore, even a highly data-parallel task contains such calculations, that task cannot be done in the accelerator cores.

In this paper, we propose a task allocation method to map multiple tasks to heterogeneous multi-core processors effectively. The proposed task allocation method changes the accelerator-incompatible operations to accelerator-compatible by using approximation methods such as CORDIC (coordinate rotation digital computer) algorithms. Then, we re-allocate such tasks to the accelerator cores to reduce the data transfers.

2. Task allocation method considering the data transfer time

In the task allocation method, we make a library that contains CORDIC algorithms for accelerator-incompatible calculations. The task allocation method starts with an initial solution by assigning data-parallel and accelerator-compatible tasks to the accelerator cores and the rest to the CPU cores. Then we estimate processing times and the data transfer times for all the tasks. Since the calculation time depends on the amount of data, the number of control steps, the number of parallel calculations and the clock
frequency, we can approximate it at high-level-synthesis. Similarly we approximate the data transfer time using the amount of data and the data transfer rate. We create a priority list that contains processing time and data transfer time information in the descending order and select the most critical processing time or the data transfer time from the top of the priority list. If it is a processing time, we consider allocating that task to a different core. If it is a data transfer time, we re-allocating the data dependent tasks either to the CPU core or to the accelerator core. When assigning tasks with accelerator-incompatible calculations, we refer the library and pick a CORDIC algorithm. After the task allocation, we evaluate the data transfer time and the calculation time. If the total processing time decreased, we accept the new assignment and update the priority list. If the total processing time increased, we reject the new assignment and remove the first entry from the priority list. Then we update the priority list and move to the next most critical processing or data transfer time. When the priority list is empty, we terminate the optimization.

3. Evaluation

3.1 HOG algorithm

We use HOG (histogram of oriented gradients) algorithm [4] to evaluate the proposed task allocation method. The HOG feature descriptor is widely used in computer vision and image processing for object detection. It contains 4 major tasks.

**Task 1: Gradient computation:** A 1-D derivative mask is applied to all the pixels in the image in horizontal and vertical directions. Equations (1) and (2) gives the horizontal and vertical derivatives respectively. The intensity of the pixel at the coordinates \( (x, y) \) is given by Eq. (1).

\[
f_x(x, y) = I(x + 1, y) - I(x - 1, y)
\]

\[
f_y(x, y) = I(x, y + 1) - I(x, y - 1)
\]

(1)

(2)

**Task 2: Orientation binning:** Each pixel within the cell casts a weighted vote given by Eq.(4) for an orientation-based histogram channel using the values found in the gradient computation. The orientation is given by Eq.(3).

\[
m(x, y) = \sqrt{f_x(x + 1, y)^2 + f_y(x - 1, y)^2}
\]

\[
\theta(x, y) = \tan^{-1} \left( \frac{f_y(x, y)}{f_x(x, y)} \right)
\]

(3)

(4)

**Task 3: Descriptor blocks:** In this task, the cells are grouped together to make larger blocks.

**Task 4: Block normalization:** If \( V_k \) is the vector of the block region and \( \epsilon \) is a very small value close to zero, the HOG descriptor \( V \) is given by Eq.(5).

\[
V = \frac{V_k}{\sqrt{||V_k||^2 + \epsilon}}
\]

(5)

### Table 1: Power and energy consumption

<table>
<thead>
<tr>
<th></th>
<th>Processing time (us)</th>
<th>Power (W)</th>
<th>Energy (Ws)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>5554</td>
<td>1.21</td>
<td>(6.72 \times 10^{-3})</td>
</tr>
<tr>
<td>CPU and accelerator (conventional method)</td>
<td>1626</td>
<td>1.30</td>
<td>(2.11 \times 10^{-3})</td>
</tr>
<tr>
<td>CPU and accelerator (proposed method)</td>
<td>330</td>
<td>1.30</td>
<td>(0.43 \times 10^{-3})</td>
</tr>
</tbody>
</table>

3.2 Experimental results

For the evaluation, we used a heterogeneous multi-core processor explained in [1]. The accelerator contains 32 processing elements that are capable of additions, multiplications, etc. However, complex operations such as square-root, division, trigonometric calculations are not available. Since tasks 2 and 4 of the HOG algorithm contain accelerator-incompatible squre-root and trigonometric operations, we map them to the CPU cores. The tasks 1 and 3 contain data-parallel processing and are mapped to the accelerator cores. Then we perform the optimization process as explained in Sec.2. In this particular example, allocating all the tasks to the accelerator core gives the largest processing speed.

Table 1 shows the comparison of processing time, power and energy consumptions. By using the conventional task allocation, we gain 70% reduction of processing time. Using the proposed task allocation method, we achieved 94% reduction of the total processing time. The power consumption has lightly increased when the accelerator core is used. However, the energy is decreased significantly due to the reduction of the processing time. In the proposed task allocation, the energy consumption reduced by 93% compared to CPU-only implementation.

4. Conclusion

We have proposed a task allocation method to reduce the processing time in heterogeneous multi-core processors by re-allocating the tasks among processor cores using approximation methods such as CORDIC algorithms. According to the experimental results with HOG feature detection algorithm, the proposed method reduces the total processing time and energy by more than 79%.

**References**


