LETTER

Dual-Core Framework: Eliminating the Bottleneck Effect of Scalar Kernels on SIMD Architectures

Yaohua WANG(a), Shuming CHEN(b), Nonmembers, Hu CHEN†, Member, Jianghua WAN†, Nonmember, Kai ZHANG†, and Sheng LIU†, Members

SUMMARY The efficiency of ubiquitous SIMD (Single Instruction Multiple Data) media processors is seriously limited by the bottleneck effect of the scalar kernels in media applications. To solve this problem, a dual-core framework, composed of a micro control unit and an instruction buffer, is proposed. This framework can dynamically decouple the scalar and vector pipelines of the original single-core SIMD architecture into two free-running cores. Thus, the bottleneck effect can be eliminated by effectively exploiting the parallelism between scalar and vector kernels. The dual-core framework achieves the best attributes of both single-core and dual-core SIMD architectures. Experimental results exhibit an average performance improvement of 33%, at an area overhead of 4.26%. What’s more, with the increase of the SIMD width, higher performance gain and lower cost can be expected.

key words: SIMD, dual-core, scalar, vector

1. Introduction

The SIMD architecture is widely used in today’s media processors. Examples include the stream processors [1] and GPUs. Processors from academic studies like MAVEN [2] and AnySP [3] also employ SIMD architectures. The SIMD architecture (Fig. 1) is generally composed of a scalar pipeline (SP) and a vector pipeline (VP). The SP is capable of the overall instruction fetching and dispatching (IFD), processing of scalar applications, and execution assistance for VP. The VP has multiple lanes running in a lock-step manner, achieving high throughput for media applications.

Although the performance improvement by using the SIMD architecture is encouraging, we should note that the overall performance is greatly limited by the non-SIMD-izable (scalar) kernels in media applications. Such kernels are relegated to execute on the SP, degrading the overall performance of SIMD architectures to that of a scalar processor [4]. With the high performance gain of SIMD-izable (vector) kernels, Amdahl’s Law will expose the scalar kernels as the overall bottleneck. What’s more, this bottleneck effect can be even more serious when the SIMD width increases.

Previous solutions for this problem mainly put their concentrations on the acceleration of scalar kernels by the VP. Ganesh [4] provided a flexible VP, in which the VP lanes could work under a feed-forward manner to exploit the instruction level parallelism and to perform operation chaining of scalar kernels. However, this method needed complex hardware configuration and compile algorithms.

To solve the bottleneck effect of scalar kernels and reduce the complex hardware and compiler support in existing studies, we propose the dual-core framework. The cornerstone of the framework is composed of a micro control unit and an instruction buffer, capable of independently assisting the execution of the VP. This capability can dynamically decouple the SP and VP of the SIMD architecture into two free-running cores. Thus, the bottleneck effect of scalar kernels can be eliminated by well exploiting the parallelism between scalar and vector kernels.

2. Problem Analysis

The parallelism between scalar and vector kernels motivates our solution for the aforementioned problem. This parallelism comes from the fact that chained scalar and vector kernels are commonly existed in media applications. Two main causes contribute to this chained characteristic. One is that this characteristic is inherently existed in media applications. Examples includes the Resource Element Demapping with FFT (Fast Fourier Transform) (FFT+RED), Resource Element Mapping with STBC (Space Time Block Code) encoding (STBC+REM), the de-interleaving kernel with the demodulation kernel (Demod+Deint) [5], and the IDCT transformation and Quantization kernels with the reordering (IQ+R). The other cause is the partially vectorizable feature of some media application kernels. These kernels are turned into vector kernels appended by scalar
and loop control operations are needed, conducting variable and type limited feature. Only move, shift, add 
conclusions can be made from Table 1. One is the simple SIMD architecture simulator described in Sect. 5.1. Two
vector kernels. These results are obtained from our baseline ble 1 lists the SP assistant operations for some representative 
assistant operations exist in most of the vector kernels. Ta-
tions are needed in scalar kernels, only a small amount of SP solve the above problem, is that although complex SP opera-
complexity and time-uncertainty.
ern media processing architectures due to their hardware problems. However, these schemes are not suitable for mod-
parallelism is with an additional scalar core. Thus, both scalar kernels can be processed concurrently on the 
parallelism. Thus, the execution time of scalar kernels can be hidden in that of the vector ones.

3. Architectural Inspiration

One possible architectural support to exploit the above parallelism is with an additional scalar core. Thus, both scalar and vector kernels can be processed concurrently on the corresponding cores. However, compared with a single-core structure, the additional scalar core is laid idle when not used for parallelism exploitation, resulting in a waste of hardware resources. Besides, such structure suffers from the communication overhead between both cores. Enhancing the SP with multithreading schemes like Simultaneous Multi-Threading (SMT) or superscalar can also solve the problem. However, these schemes are not suitable for modern media processing architectures due to their hardware complexity and time-uncertainty.

An insightful observation, which inspires us to well solve the above problem, is that although complex SP operations are needed in scalar kernels, only a small amount of SP assistant operations exist in most of the vector kernels. Table 1 lists the SP assistant operations for some representative vector kernels. These results are obtained from our baseline SIMD architecture simulator described in Sect. 5.1. Two conclusions can be made from Table 1. One is the simple and type limited feature. Only move, shift, add/subtraction and loop control operations are needed, conducting variable initialization and loop-control operations. The other is the small amount of operations. During the execution of a vector kernel, the SP is only busy at the beginning and the end of iteration, constituting around 10% of the whole vector kernel’s execution time.

The reason for SP assistant operations is that in order to take full advantage of abundant on-chip computation resources in SIMD architectures, various techniques have been employed to achieve a well-aligned computation pattern with simple control-flows and regular memory accesses for vector kernels, reducing a large amount of SP assistant operations. For instance, control-flow handling operations are cut down to only loop control instructions, and memory address generation operations are eliminated by the auto-increase/decrease memory accesses.

Although only a small amount of SP assistant operations are needed in a vector kernel, without additional hardware support, the idleness of SP is difficult to be utilized by a software only method. As both the SP and VP work under an identical control-flow. The different control-flows of scalar and vector kernels make it difficult, and sometimes even impossible to schedule a scalar kernel in the execution context of a vector one.

To take full advantage of the observations above, we propose a dual-core framework, which can dynamically decouple the SP and VP of the SIMD architecture into two cores. Thus, the parallelism between scalar and vector kernels can be well exploited with the efficiency of the traditional single-core SIMD architecture maintained.

4. The Dual-Core Framework

The block diagram of the dual-core framework architecture is shown in Fig. 3. It is designed around a micro control unit (MC) and an instruction buffer (I-Buff). With the help of this dual-core framework, the entire architecture can dynamically change between the single-core and dual-core modes. In the single-core mode, the IFD issues instructions to both SP and VP. In the dual-core mode, IFD only issues instructions to SP, while the MC issues instructions from the I-Buff to VP.

4.1 Main Hardware Components

The main functions of MC include two parts. One is the capability of SP assistant functions, and the other is the conduction of instruction saving and issuing process of the I-Buff. MC adopts a simple scalar pipeline structure, supporting only simple SP assistant operations as listed in Table 1. The function units are fed by a 16-entry local register file. In the single core mode, MC saves vector kernel’s instructions into the I-Buff at the same time when instructions are issued from SP in the vector kernel’s first loop iteration. Thus, the instruction saving overhead can be hidden in the execution process. After instructions of the loop body have been saved in the I-Buff, the dual-core execution mode can start. In the dual-core mode, the instruction path from IFD to VP is dis-
Table 1 SP assistant operations for vector kernels.

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Op</th>
<th>move</th>
<th>add sub</th>
<th>shift</th>
<th>loop control</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT</td>
<td>0.91%</td>
<td>4.49%</td>
<td>0.36%</td>
<td>2.91%</td>
<td>8.67%</td>
<td></td>
</tr>
<tr>
<td>IDCT</td>
<td>7.73%</td>
<td>1.23%</td>
<td></td>
<td>4.12%</td>
<td>13.08%</td>
<td></td>
</tr>
<tr>
<td>Quantization</td>
<td>6.62%</td>
<td>1.56%</td>
<td></td>
<td>1.50%</td>
<td>9.74%</td>
<td></td>
</tr>
<tr>
<td>STBC</td>
<td>3.66%</td>
<td>1.61%</td>
<td></td>
<td>0.2%</td>
<td>5.47%</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 3 The dual-core framework architecture.

Fig. 4 The pipeline execution on the dual-core framework.

aborted, and MC issues instructions from the I-Buff to both VP and the MC itself.

The I-Buff is used to save instructions of vector kernels. The loop-structure of vector kernels and the small amount of SP assistant operations confirm the efficiency of buffering vector kernels’ instructions. We adopt a fixed-length VLIW (Very Long Instruction Word) structure for the I-Buff. As shown in Fig. 3, each row of the I-Buff contains an instruction packet, consisting of instructions for both the MC and the VP. Only one instruction slot is reserved for MC due to the small amount of SP assistant operations. The fixed-length VLIW structure provides a one-to-one correspondence between the function units of VP and the instruction slots in the I-Buff, leading to simplified instruction saving and issuing processes. The proper depth of I-Buff can be obtained from the profiling in the target application domain, as that did in the stream processor [1]. Here, we set it to be 512, which is enough for the vector kernels evaluated in this paper.

4.2 The Pipeline Execution Scheme

Four special instructions, including \textit{WRTEN}, \textit{WRTDE}, \textit{DIVIDE} and \textit{SYNCH}, are introduced to efficiently support the pipeline execution for chained scalar and vector kernels. These instructions are executed in the SP. \textit{WRTEN} is used to enable the instruction saving process, which is then disabled by \textit{WRTDE}. \textit{DIVIDE} initiates the dual-core execution mode by disabling the instruction path from the IFD to VP and enabling the instruction issuing process of I-Buff. \textit{SYNCH} recovers the single core mode.

With the help of these proposed instructions, the pipeline execution of chained scalar and vector kernels can be done efficiently. Figure 4 illustrates the code example for STBC+REM (shown in Fig. 2 (b)). In the prologue phase, \textit{WRTEN} starts the instruction saving process, and then \textit{WRTDE} disables the saving process after instructions have been saved into the I-Buff. In the loop body phase, \textit{DIVIDE} starts the dual-core execution mode at the beginning of each loop. Thus, STBC and REM can be processed concurrently. Only instructions of REM are needed in the loop body, as instructions of STBC are stored in the I-Buff. \textit{SYNCH} synchronizes both kernels and recovers the entire architecture to the single core mode at the end of STBC. The epilogue phase is done in the single core mode.

4.3 Exception Handling

In modern SIMD architectures [2], [3], to achieve high performance while meeting the need for real time execution, immediate exception handling is usually not supported (except for exceptions caused by external interrupts) because of the large performance loss and time uncertainty. Alternatively, exceptions are usually handled by storing the exception information in flag bits, and then software inserts trap barrier instructions to check the flag bits as needed.

The proposed dual-core framework does not affect this flag-bits based exception handling. The only limitation is that for time certainty and simplicity, VP is undisturbed when running in the dual-mode. Thus, exceptions caused by the external interrupts are not allowed to have instructions executed in the VP. However, as most of the external interrupt handler program contains only SP instructions, this limitation will not be a big problem.

4.4 Hardware Cost

When adopting the dual-core framework in existing SIMD architectures, simple decoding logic for the introduced instructions, and two additional instruction paths are needed beside the MC and I-Buff. However, the additional cost of the decoding logic and instruction paths can be omitted compared with that of the MC and I-Buff. Moreover, in either the single-core and dual-core modes, the MC and I-Buff are in parallel with VP or SP, and will not affect the critical path of the original processor. Thus, the overall frequency of the SIMD architecture will not be degraded.

We have implemented the MC and I-Buff of the dual-core framework in verilog, and Synopsys Design Compiler is used to synthesize our design in TSMC 65 nm technology at 700 MHz. Table 2 shows the area breakdown. With the instruction width set equal to that in AnySP [3], the whole overhead of the framework is about 0.14 mm².
Table 2 The hardware cost of the dual-core framework.

<table>
<thead>
<tr>
<th>Component</th>
<th>Parameters</th>
<th>Area (mm²)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC</td>
<td>4 function units 16-entry reg</td>
<td>0.011835</td>
</tr>
<tr>
<td>I-Buff</td>
<td>512<em>32</em>9 bits</td>
<td>0.128451</td>
</tr>
</tbody>
</table>

With 4 SIMD cores in AnySP, the overall area overhead becomes 0.56 mm², which is 4.26% of the total area of AnySP-like traditional SIMD architectures (the area of AnySP is 13.14 mm²).

4.5 Discussion

Compared with the mighty-morphing method [4], the overhead of our dual-core framework is much smaller because of the absence of configurable connections among VP lanes and the complex compiler support. However, as the dual-core framework needs parallel kernels to hide the scalar ones. It may happen that some scalar kernels could not be well hidden. Thus, the dual-core framework and the mighty-morphing method complement each other. The dual-core framework can help to hide scalar kernels requiring complex hardware configurations, leading to much simpler hardware configurations for the mighty-morphing method. In the same way, the mighty-morphing method can help to accelerate those scalar kernels that could not be well hidden.

5. Evaluation

5.1 Methodology

We build a cycle-accurate simulator to evaluate the performance of the dual-core framework. Manually optimized assemble code is used as the input of the simulator. The baseline SIMD architecture is similar to the one shown in Fig. 1. The number of VP lanes is 16, with each lane contains four function units: ALU, L/S and two MACs. The SP contains four function units: branch, MAC, ALU, and L/S. Representative media application kernels discussed in Sect. 2 are listed in Table 3.

Table 3 Benchmark description.

<table>
<thead>
<tr>
<th>kernels</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFT+RED</td>
<td>2048 radix-2 FFT 1320*14 resource grid</td>
</tr>
<tr>
<td>STBC+REM</td>
<td>4T4R STBC 1320*14 resource grid</td>
</tr>
<tr>
<td>Demode+Deint</td>
<td>64QAM</td>
</tr>
<tr>
<td>IQ+R</td>
<td>16<em>16 MBs with size 4</em>4</td>
</tr>
<tr>
<td>VITERBI</td>
<td>length: 9 Rate: 1/3 iteration: 8</td>
</tr>
<tr>
<td>TURBO</td>
<td>length: 9 Rate: 1/3 iteration: 8</td>
</tr>
<tr>
<td>ME</td>
<td>area: 32<em>32 macro block:16</em>16</td>
</tr>
</tbody>
</table>

5.2 The Overall Performance Gain

Figure 5 shows the overall speedup. The baseline performance is obtained by processing the scalar and vector kernels sequentially. An average performance gain of 33% can be achieved. The performance gain is mainly affected by two factors. One is the equality of execution time between scalar and vector kernels. The more equal they are, the much higher performance gain can be achieved. The execution time of REM and R is about 75% and 80% of the corresponding vector kernels, leading to a high performance gain. The other factor is the size of the pro/epilogue for the pipeline execution. The prologue for FFT+RED is relatively large, due to the large data block grain of FFT when pipelined with RED. Thus, the performance gain is a little small. The same story occurs in VITERBI and TURBO. However, in media applications, there are a large number of kernels, so that the pro/epilogue part of pipeline execution can always be overlapped.

5.3 The SIMD Width Effect

To reveal the effect of SIMD width on the performance of the dual-core framework, we vary the SIMD width from 8 to 64. The results are shown in Fig. 6. The performance gain becomes higher with the increase of the SIMD width. The reason is that a wider SIMD architecture leads to a greater amount of speedup for vector kernels, resulting in a relatively equal execution time between the scalar and vector kernels.

It is interesting that for STBC+REM and IQ+R, the performance gain for a larger SIMD width becomes smaller. This is because that for a larger SIMD width, ‘REM’ and ‘R’ consume a larger proportion of execution time than the corresponding vector kernels. Thus, their execution time cannot be fully hidden in the vector ones. This phenomenon indicates that a high performance SP is of benefit to future wider SIMD architectures.

6. Conclusion

This paper presents a dual-core framework, capable of dynamically decoupling the SP and VP of the SIMD architecture into two free-running cores. Thus, the scalar kernel’s
bottleneck effect can be eliminated by exploiting the parallelism between scalar and vector kernels. The evaluations demonstrate that the dual-core framework makes the SIMD architecture more efficient and flexible.

References


