OpenCL Altera SDK v.14.0 vs. v. 13.1 Benchmarks Study

ABEDALMUHDI ALMOMANY* and AMIN JARRAH

Computer Engineering Department, Yarmouk University, Irbid, Jordan.

Abstract
Alterna SDK for OpenCL allows programmers to write a simple code in OpenCL and abstracts all Field programmable gate array (FPGA) design complexity. The kernels are synthesized to equivalent circuits using the FPGA hardware resources Adaptive logic modules (ALMs), DSPs and Memory blocks. In this study, we developed a set of fifteen different benchmarks, each of which has its own characteristics. Benchmarks include with/without loop unrolling, have/have not atomic operations, have one/multiple kernels per single file, and in addition to one/more of these characteristics are combined. Altera OpenCL v14.0 adds more features compared with previous versions. A set of parameters chosen to compare the two OpenCL SDK versions Logic utilization (in ALMs), total registers, RAM Blocks, total block memory bits, and clock frequency.

Introduction
OpenCL stands for Open Computing Language, which is an open framework for parallel programming executed across heterogeneous platforms CPUs, GPUs and DSPs. OpenCL programming model consists of two programs; first, host program, which is usually written in C/C++, and it is responsible for loading the OpenCL programs, memory management, data transfer and errors checking. Second program is the device code, which is written in OpenCL, and can be run on the available devices such as GPUs, DSPs, or FPGAs.

In OpenCL, kernel could be executed by a large number of work-items (threads). Work-items are organized in one, two or three dimensions, and are divided into blocks which can be multi-dimensions. Each block is called a workgroup. The size of a workgroup can be up to 1024 or 2048 work-items depending on device capability. All work-items inside the workgroup can be synchronized using barrier. However, synchronization cannot be between workgroups, and they could be executed in any order.

The Altera SDK for OpenCL allows the programmer to implement parallel algorithms on FPGA with a high level of hardware abstraction. The Altera offline compiler (AOC) is used to generate the Altera executable file, which can be run on the FPGA (DE5 in this study). Each kernel is synthesized to an equivalent circuit on the FPGA board, and
each circuit contains a set of hardware resources. FPGAs implement parallel algorithms using pipelining architecture where input data passes through a sequence of stages.\textsuperscript{4,5} FPGA main resources include Adaptive logic modules (ALM), digital signal processing (DSP) and memory blocks.

AOC is used to create a hardware configuration file. Some parameters can be combined for the optimization purpose. Compilation process is very length, which can range anywhere between minutes and several days. In the set of benchmarks here, the compilation time ranges between one hour and few minutes up to six hours and few minutes.

The Altera SDK 14.0 has been developed to include new features, such as supporting hard floating points, channel extensions, supporting new types (float 3) and other features.\textsuperscript{5} Our motivation behind this study is to show how these new features could affect the performance by compiling and running set of benchmarks.

FPGAs are widely used to improve the performance in several scientific applications.\textsuperscript{7-26} The FPGA device used here is Stratix V ALM is developed to implement most of function efficiently. Each ALM contains a look-up table (eight inputs), two dedicated adders and four dedicated registers. The LUT can implement any 6-input logic functions and a number of 7-input functions. It can also be used as two separated LUTs for efficient using. The block diagram for ALM is shown in figure-1.\textsuperscript{6}

**Experimtial Setup Environments**

- Linux 2.6.32-504.1.3.el6.x86_64
- Altera SDK, 64-Bit Offline Compiler Ver. 14.0
- Altera SDK, 64-Bit Offline Compiler Ver. 13.1
- gcc version 4.4.7
- DE5 Board (StratixV,Dev5SGXEA7N2F45C2)

**Experiment and Results Discussion**

Several studies handle the issue of comparing different compilers.\textsuperscript{3,4} To compare the two Altera SDK versions, a set of fifteen benchmarks were developed for comparison purpose. These benchmarks are varied in their characteristics as follows none, one, or more atomic operations, with/without loop unrolling, single/multiple kernels per file.

The benchmarks written can be classified as pure memory access, where the whole kernel is written using reads or writes memory operations. The read/write operations could be atomic or non-atomic, using same or different atomic operation. “atomic add” and atomic exchange are used in this study. The other class is consisted of a set of arithmetic operations on floating points. These operations include four main operations (addition, subtraction, division, and multiplication). The OpenCL kernels can repeat the same code many times, where loop unrolling is used in some kernels. The same kernels run again but without loop unrolling in other benchmarks. The last thing tested using these benchmarks is repeating the same kernel in the file up to seven times, or using more than one kernel with different characteristics. In summary, a set of fifteen benchmarks summaries all of the above attributes. A set of parameters are concerned here: logic utilization in ALMs, RAM blocks, total memory bits, clock frequency, total registers and compile time. Other parameters might be added here are size of configuration and backup files created. Our results show that the size of the files created by Altera 14.0 is less by 400MBs.

The FPGA device used in the experiments contains 234,720 ALMs, 256 DSP Blocks, 52,428,800 block memory bits and 2,560 RAM Blocks.
### Table 1: Altera 13.0 Benchmarks results

<table>
<thead>
<tr>
<th>Altera 13.0</th>
<th>Bench1</th>
<th>Bench2</th>
<th>Bench3</th>
<th>Bench4</th>
<th>Bench5</th>
<th>Bench6</th>
<th>Bench7</th>
<th>Bench8</th>
<th>Bench9</th>
<th>Bench10</th>
<th>Bench11</th>
<th>Bench12</th>
<th>Bench13</th>
<th>Bench14</th>
<th>Bench15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic utilization (ALMS)</td>
<td>70%</td>
<td>66%</td>
<td>66%</td>
<td>67%</td>
<td>70%</td>
<td>70%</td>
<td>66%</td>
<td>16%</td>
<td>25%</td>
<td>65%</td>
<td>38%</td>
<td>20%</td>
<td>23%</td>
<td>26%</td>
<td>29%</td>
</tr>
<tr>
<td>Total registers</td>
<td>317785</td>
<td>298505</td>
<td>298246</td>
<td>300935</td>
<td>318041</td>
<td>318501</td>
<td>300312</td>
<td>86153</td>
<td>305756</td>
<td>143413</td>
<td>72397</td>
<td>84439</td>
<td>97485</td>
<td>106650</td>
<td></td>
</tr>
<tr>
<td>RAM Blocks Percentage</td>
<td>71.3%</td>
<td>64.3%</td>
<td>64.3%</td>
<td>65.4%</td>
<td>72%</td>
<td>71.3%</td>
<td>64.3%</td>
<td>11%</td>
<td>17.9%</td>
<td>18.6%</td>
<td>23.5%</td>
<td>14.1%</td>
<td>11.4%</td>
<td>20%</td>
<td>21.7%</td>
</tr>
<tr>
<td>Total Block memory Bits</td>
<td>12%</td>
<td>11%</td>
<td>11%</td>
<td>12%</td>
<td>12%</td>
<td>12%</td>
<td>3%</td>
<td>4%</td>
<td>18%</td>
<td>5%</td>
<td>3%</td>
<td>3%</td>
<td>4%</td>
<td>4%</td>
<td>4%</td>
</tr>
<tr>
<td>Actual Clock frequency</td>
<td>189</td>
<td>203</td>
<td>200</td>
<td>193</td>
<td>194</td>
<td>193</td>
<td>195</td>
<td>305</td>
<td>246</td>
<td>187</td>
<td>206</td>
<td>211</td>
<td>185</td>
<td>194</td>
<td>203</td>
</tr>
<tr>
<td>compile Time in minutes</td>
<td>310</td>
<td>296</td>
<td>290</td>
<td>303</td>
<td>305</td>
<td>299</td>
<td>294</td>
<td>63</td>
<td>100</td>
<td>344</td>
<td>153</td>
<td>88</td>
<td>101</td>
<td>112</td>
<td>114</td>
</tr>
</tbody>
</table>

### Table 2: Altera 14.0 Benchmarks results

<table>
<thead>
<tr>
<th>Altera 14.0</th>
<th>Bench1</th>
<th>Bench2</th>
<th>Bench3</th>
<th>Bench4</th>
<th>Bench5</th>
<th>Bench6</th>
<th>Bench7</th>
<th>Bench8</th>
<th>Bench9</th>
<th>Bench10</th>
<th>Bench11</th>
<th>Bench12</th>
<th>Bench13</th>
<th>Bench14</th>
<th>Bench15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic utilization (ALMS)</td>
<td>59%</td>
<td>59%</td>
<td>59%</td>
<td>59%</td>
<td>59%</td>
<td>59%</td>
<td>59%</td>
<td>18%</td>
<td>26%</td>
<td>69%</td>
<td>40%</td>
<td>20%</td>
<td>21%</td>
<td>23%</td>
<td>24%</td>
</tr>
<tr>
<td>Total registers</td>
<td>253721</td>
<td>241166</td>
<td>241166</td>
<td>247225</td>
<td>255229</td>
<td>255229</td>
<td>242248</td>
<td>55886</td>
<td>90962</td>
<td>308050</td>
<td>160268</td>
<td>71354</td>
<td>71532</td>
<td>79783</td>
<td>88015</td>
</tr>
<tr>
<td>RAM Blocks Percentage</td>
<td>54.7%</td>
<td>37.6%</td>
<td>37.6%</td>
<td>42.7%</td>
<td>56.9%</td>
<td>56.9%</td>
<td>38.8%</td>
<td>11%</td>
<td>17.1%</td>
<td>74%</td>
<td>21.6%</td>
<td>14.7%</td>
<td>16.7%</td>
<td>18.4%</td>
<td>20%</td>
</tr>
<tr>
<td>Total Block memory Bits</td>
<td>10%</td>
<td>9%</td>
<td>9%</td>
<td>10%</td>
<td>11%</td>
<td>11%</td>
<td>9%</td>
<td>3%</td>
<td>3%</td>
<td>18%</td>
<td>5%</td>
<td>3%</td>
<td>5%</td>
<td>6%</td>
<td>6%</td>
</tr>
<tr>
<td>Actual Clock frequency</td>
<td>171.17</td>
<td>186.18</td>
<td>186.18</td>
<td>197.17</td>
<td>174.21</td>
<td>174.20</td>
<td>204.26</td>
<td>266.22</td>
<td>220.18</td>
<td>21113</td>
<td>236.24</td>
<td>241.22</td>
<td>226.23</td>
<td>231.23</td>
<td></td>
</tr>
<tr>
<td>compile Time in minutes</td>
<td>262</td>
<td>256.25</td>
<td>257.24</td>
<td>249.25</td>
<td>257.25</td>
<td>250.24</td>
<td>247.69</td>
<td>92.338</td>
<td>148.83</td>
<td>85.125</td>
<td>99.23</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Examining both tables, it is clear that the Altera SDK 14.0 shows better optimization of resources, and requires less compilation time. On the other hand, the clock frequency may not be enhanced, but may be decreased. Dividing the values in Table II by the corresponding values in Table I and averaging each row will generate the results shown in Table III. This gives a comparison between the two versions considering the parameters mentioned above. Taking the average effects of all parameters, every parameter is normalized to the Altera SDK 13.1 corresponding parameter.

<table>
<thead>
<tr>
<th>Table 3: Comparison Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Utilization</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>Altera 13.1</td>
</tr>
<tr>
<td>Altera 14.0</td>
</tr>
</tbody>
</table>

Fig. 2: Altera SDK 14.0 vs. Altera SDK 13.1 Comparison results

**Conclusion**
Our study shows that using the Altera SDK 14.0 for the previous benchmarks provides better resources utilization. We need fewer resources compared to the Altera SDK 13.0. Although the clock speed may decrease or increase, the changes is insignificant. We recommend using Altera SDK14.0 instead of Altera SDK 13.0. In future paper, the comparison will handle the most recent Intel FPGA compilers.

**Acknowledgement**
Nill

**Funding**
The author(s) received no financial support for the research, authorship, and/or publication of this article.

**Conflict of Interest**
This manuscript has not been submitted to, nor is under review at, another journal or other publishing venue. The authors certify that they have NO affiliations with or involvement in any organization or entity with any financial interest (such as honoraria; educational grants, participation in speakers’ bureaus, membership, employment, consultancies, stock ownership, or other equity interest, and expert testimony or patent-licensing arrangements), or non-financial interest (such as personal or professional relationships, affiliations, knowledge or beliefs) in the subject matter or materials discussed in this manuscript.
References


and Computers (pp. 1839-1842). IEEE.


