# Analysis of Fixed, Reconfigurable, and Hybrid Devices with Computational, Memory, I/O, & Realizable-Utilization Metrics



JUSTIN RICHARDSON, ALAN GEORGE, KEVIN CHENG, and HERMAN LAM,
NSF Center for High-Performance Reconfigurable Computing (CHREC) at the University of Florida

The modern processor landscape is a varied and diverse community. As such, developers need a way to quickly and fairly compare various devices for use with particular applications. This article expands the authors' previously published computational-density metrics and presents an analysis of a new generation of various device architectures, including CPU, DSP, FPGA, GPU, and hybrid architectures. Also, new memory metrics are added to expand the existing suite of metrics to characterize the memory resources on various processing devices. Finally, a new relational metric, *realizable utilization (RU)*, is introduced, which quantifies the fraction of the computational density metric that an application achieves within an individual implementation. The RU metric can be used to provide valuable feedback to application developers and architecture designers by highlighting the upper bound on specific application optimization and providing a quantifiable measure of theoretical and realizable performance. Overall, the analysis in this article quantifies the performance tradeoffs among the architectures studied, the memory characteristics of different device types, and the efficiency of device architectures.

CCS Concepts: • **Hardware → Hardware accelerators**; **Emerging architectures**; *Reconfigurable logic applications*; *Emerging tools and methodologies*

Additional Key Words and Phrases: Device characterization, benchmarking, performance, device studies, comparative analysis

**ACM Reference Format:**
Justin Richardson, Alan George, Kevin Cheng, and Herman Lam. 2016. Analysis of fixed, reconfigurable, and hybrid devices with computational, memory, I/O, & realizable-utilization metrics. ACM Trans. Reconfigurable Technol. Syst. 10, 1, Article 2 (September 2016), 21 pages.
DOI: http://dx.doi.org/10.1145/2888401

## 1. INTRODUCTION

The ever-changing landscape of computational technologies has created a need for methodologies by which diverse processing architectures can be quickly and objectively compared. Previously published methods, including our own, based on computational device metrics have been effective in performing such comparisons [DeHon 1996; Williams et al. 2011; Milluzzi et al. 2014; Sohi and Franklin 1991; Saulsbury et al. 1996; Burger et al. 1996]. Using this approach, a modern processing device, regardless of whether it is fixed-logic (e.g., Central Processing Unit, Digital Signal Processor, Graphics Processing Unit), reconfigurable-logic (e.g., Field Programmable Gate Array, Complex Programmable Logic Device), or hybrid (e.g., CPU/FPGA, CPU/DSP,

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grants No. EEC-0642422 and No. IIP-1161022.
Authors' addresses: J. Richardson, 2033 Mowry RD Room 119, PO BOX 103610, Gainesville, FL 32611-0001; A. George, K. Cheng, and H. Lam, NSF CHREC Center, ECE Department, University of Florida, Room 320, Larsen Hall, 968 Center Drive, POB 116200, Gainesville, FL 32611-6200.

DOI: http://dx.doi.org/10.1145/2888401

CPU/GPU), can be abstractly characterized using key device metrics and thus be compared objectively.

This article expands the authors' previously published work in three ways. First, using metrics that we established in Williams et al. [2011], this article presents an analysis of a new generation of various device architectures. Second, this article expands our existing suite of metrics with new metrics to characterize the memory resources on various processing devices. Finally, a third contribution of this article is to introduce a new relational metric, *realizable utilization (RU)*, which quantifies the fraction of the computational density metric (i.e., upper bound) that an application achieves within an individual implementation. Focusing on these concepts, a brief outline of each section is presented in the following paragraphs.

In this article, a large suite of fixed-logic, reconfigurable-logic, and hybrid devices is analyzed based on their computational density and computational density per watt. These devices were chosen as representative samples of differing processing options with a variety of architectures. The devices studied span a wide variety of feature sizes, peripheral combinations, and iterative generations to provide the most relatable results possible. The computational density (CD) of a device characterizes the computational capacity of the available processing cores on a device. Computational density per watt (CD/W) is the power-aware version of this metric and normalizes the computational density by power consumption. In Section 2, these device metrics will be reviewed in more detail and the computational metrics will be used to analyze a large suite of new generations of CPU, DSP, FPGA, GPU, and hybrid devices. The results provide insight into the strengths and weaknesses of different device architectures in relation to theoretical computational performance. From the CD and CD/W analysis, this article will show the different performance tradeoffs among device architectures, operation precisions, and power efficiencies.

In addition to computational capacity, the bandwidth of external memory units is a crucial performance bottleneck. In order to capture the performance of both memory and input/output (I/O) peripherals, the external memory bandwidth (EMB) and the input-output bandwidth (IOB) metrics are introduced for characterizing and comparing devices in regards to data movement. EMB and IOB, which highlight memory resources, will be defined in more detail in Section 3 and will be used to analyze the same suite of devices. The results from this section will showcase the relative strengths of devices with high-speed memories in contrast to devices with more generic I/O capabilities.

Finally, device metrics provide a first-order analysis, effectively providing an *upper bound* of a device's capability. How much of the device's upper-bound capability can be used in a particular application, however, is determined by many factors, including application characteristics, design tools, and user experience. To explore this relationship, Section 4 introduces and defines the realizable-utilization metric. The authors use *RU* to show how close real-world applications can get to the upper bound found with the metrics. Prior to benchmarking investment, RU results from a *literature study*, focused on GPUs, showcases the decreasing application efficiency as the computational density of new devices increases. RU *benchmarking* on a large set of kernels and devices provides insight into performance that is gained using hand-coded intrinsics or optimized libraries versus benchmarking kernels coded for portability. In summary, the RU metric can be used to provide valuable feedback to application developers and architecture designers by highlighting the upper bound on specific application optimization and providing a quantifiable measure of theoretical and realizable performance. Section 5 concludes this article with a final summary of insights and directions for future work.

## 2. CD AND CD/W

In this section, we review two of our previously published metrics that characterize a device's computational capacity, CD and CD/W. Following the review, Sections 2.2 and 2.3 show the results and give analysis of the top-scoring new devices studied. This analysis provides insights into how a device's architecture can be characterized and compared with other device architectures available on the market.

### 2.1. Review of CD and CD/W

Our previous work in Williams et al. [2011], reviewed here, introduced CD, a metric that was used to determine the computational capability of a device and was used to compare devices both within and between architectural categories using operations of varying data types such as floating-point, integer, and bit-level. This methodology also allowed for varying operations such as addition, multiplication, multiply-accumulate, and so on, and the ratio between each instruction type and the data precision (e.g., 16-bit, 32-bit, 64-bit) could easily be adjusted as desired. This flexibility allowed for analysis of data operations of any size the hardware could support. CD results were presented and used to evaluate the computational performance of the two broad categories of processing devices, fixed-logic and reconfigurable-logic. Fixed-logic devices have a fixed hardware structure that cannot be changed after fabrication (e.g., CPUs, DSPs, GPUs). Reconfigurable-logic devices can change their logical hardware structure after fabrication to adapt to changing problem requirements (e.g., FPGAs). The reader may refer to Williams et al. [2011] for a more detailed discussion on the reconfigurability factors that were used to classify a device.

In our previous work and in this new study, we focused on addition and multiplication instructions for data types ranging from 16-bit integer (i.e., Int16) to 64-bit double-precision floating point (i.e., DPFP). During this study, we maximized the number of parallel operations while keeping the number of additions and multiplications equal; however, the methodology allows for other operations and mixes as desired. The units for CD are operations per second (OPS) and, when calculating the number of parallel operations supported by a device, we considered a hardware-supported, multiply accumulate as only one operation. We count fused multiply-accumulate operations as a single operation because otherwise it introduces a data dependency between the multiplications and additions. For comparability and space, we have limited our analysis to separate addition and multiplication instructions, but the same methodology would work for multiply-accumulate. The CD used in this study is memory sustainable, where we assume the register to processing device bandwidth is not a limiting factor, and the memory-sustainable limitation comes from the closest level of memory to the computation device. For example, some of the common limiting memory structures include the following: L1 caches for most CPUs and DSPs, shader memories for GPUs, and on-chip BRAMs for FPGAs.

As an example of CD for fixed-logic devices, Equations (1) and (2) show the CD calculations for a 32-bit integer (Int32) analysis with a 50% add-multiply split for the AMD Trinity A10-6800K APU. Since this is a hybrid device, the contributions from both the CPU and GPU halves were combined. For the CPU side, the operating frequency of the device was multiplied by both the number of operating cores and the sum of all available processing elements running Int32 additions and multiplications, as shown in Equation (1). To maintain memory sustainability, the number of operations that could be processed was limited by the incoming data bus width. This limitation represents the memory subsystems' ability to keep the operational units filled with new data. Thus, the data bus width (128-bit) of each unit was divided by the data type size (32-bit) to determine the number of instructions that could be supported by the memory

architectures at the same time,

$$CD_{Int32_{CPU}} = 4.4\ GHz \times 4 \times \sum_{i=1}^{4} \frac{128}{32} = 281.6\ GOPS. \quad (1)$$

The GPU side, Equation (2), was similar, with the key differences being the operating frequency of the GPU (0.844GHz) and the number of available computational units (16). These computational units are connected to a 128-bit bus on which 32-bit data can be packed to provide memory sustainability. Together, these equations yielded a total device CD of 605.7 Giga-Operations Per Second (GOPS). In this example, the GPU's single-instruction, multiple-data (SIMD) processing units contributed significantly to the overall computational capability of the device,

$$CD_{Int32_{GPU}} = 0.844\ GHz \times 6 \times \sum_{i=1}^{16} \frac{128}{32} = 324.1\ GOPS. \quad (2)$$

For reconfigurable-logic devices such as FPGAs, Int32 CD was determined using achievable frequency and the number of parallel operations of fully utilized DSP resources and logic fabric. A single integer core for both addition and multiplication was instantiated on an FPGA using vendor IP cores. Each core was fully pipelined and the resource utilization, along with the maximum achievable frequency, was determined from vendor tools. This information allowed the number of simultaneous cores that could be instantiated on a device to be determined by using all available DSP and logic resources while reserving 15% overhead for steering logic and I/O interfacing. Again, for our analysis, only addition and multiplication operations were considered and balanced, but other hardware-supported operations and mixes of operations could easily be studied. The number of parallel operations was multiplied by the maximum achievable frequency, which was limited to the lowest of multiplication and addition. Based on the amount of available on-chip memory resources, the number of available parallel operations in the CD calculation was limited. This limitation was enforced in order to account for memory bandwidth of on-chip RAM resources for data buffering, which could have a limiting effect on the peak CD. The on-chip memory must allocate two operands per operation for memory-sustainable CD. This provision ensured that the number of parallel operations a device could support was limited by the capability of the internal memory structure to provide data for each parallel operation.

As processing devices grow in computational capacity and power consumption, device efficiency is becoming a major concern. Our methodology introduced a power-efficiency metric to quantify this important information in the form of CD/W. This metric is calculated by taking the CD and dividing by the power consumption at the level of device utilization used to compute the CD. For fixed-logic processors, the maximum thermal design power (TDP) is used for device comparability. For reconfigurable-logic processors, vendor tools, most notably the power estimator worksheets, are used to estimate power usage of the device, as in Williams et al. [2008a, 2008b, 2011] and much like TDP on fixed devices, with a worst-case toggle rate (100%) and default temperatures.

This worst-case toggle rate was chosen for two major reasons. First, to make the power comparison with fixed architectures as direct as possible, our choice was to look at design parameters that create the upper bound of the power consumption of reconfigurable devices. This choice represented the most diverse and yet still comparable point on the power consumption curve of reconfigurable devices. Thus, the use of both worst-case reconfigurable power usage and TDP make a consistent comparison between the two major types of device categories. Second, the choice of toggle rate allowed our metrics to remain application agnostic. By leaning on the conservative side, our metrics will encompass almost all realistic and conceivable applications and

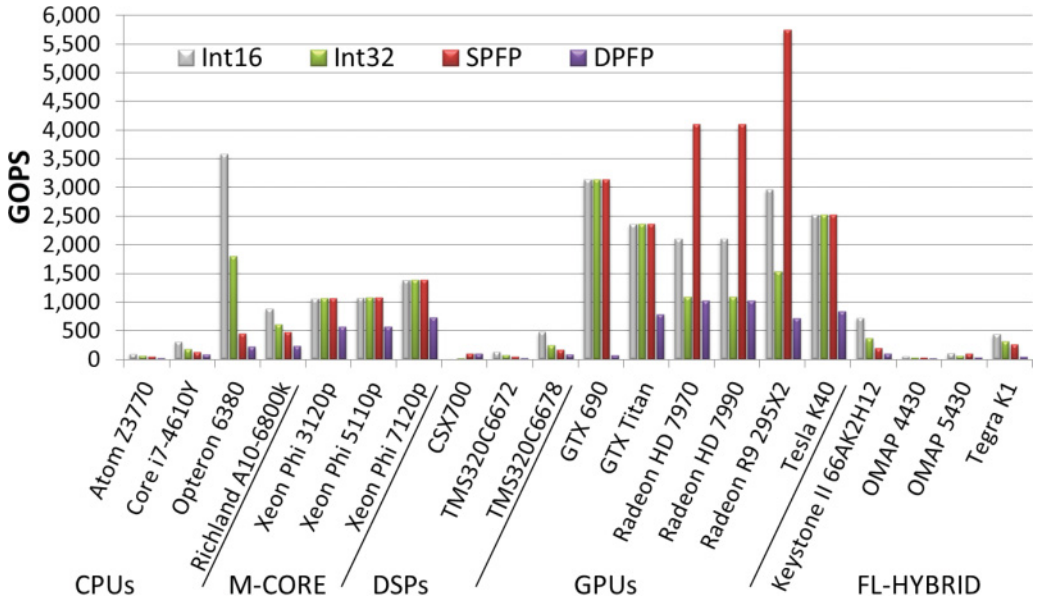


Fig. 1. CD for select fixed-logic processors.

does not require the analyst to know information about the target application design to calculate metrics.

In addition, our metrics consider Block Random Access Memory (BRAM) units all enabled with both concurrent reads and writes while running in simple dual-port mode. Dynamic power is assumed to scale linearly with resource utilization if more detailed information is unavailable. For an AMD Trinity A10-6800K, for example, the Int32 CD (605.7 GOPS) is divided by the TDP (100 watts), yielding an Int32 CD/W of 6.06 GOPS/W. Likewise, for a Xilinx SX475T, the Int32 CD (314.94 GOPS) is divided by the estimated power (20.54 watts) to give an Int32 CD/W of 15.33 GOPS/W. For more details on CD and CD/W and their applications, the reader may refer to Williams et al. [2008a, 2008b, 2011]. These metrics have been applied to a broad range of modern processors with new results and analysis presented in the following subsections.

### 2.2. CD Results

During this study, we analyzed the CD and CD/W values of 130 newly studied devices, with feature size less than or equal to 90nm, spanning fixed and reconfigurable-logic architectures with their hybrids. First, this subsection will overview the most interesting CD results and analysis from the 81 fixed-logic, 32 reconfigurable-logic, and 17 hybrid devices. Similarly, Section 2.3 will highlight the CD/W results and analysis for the same set of devices.

Due to the wealth of data, we have filtered the fixed-logic results into a subset based on the top-scoring devices in either CD or CD/W. We grouped the fixed-logic devices into five major categories: CPUs, GPUs, DSPs, many-core (M-CORE), and fixed-logic hybrids (FL-HYBRID). For this study, we focused on Int16, Int32, single-precision floating point (SPFP), and DPFP data types for both addition and multiplication operations. Our operation mix was set at 50% adds and 50% multiplies. For a full list of all the fixed-logic devices studied, with their respective CD values, please see Table I in the appendix. Figure 1 shows the CD results for the fixed-logic processors in each category with the bars representing different operation types.

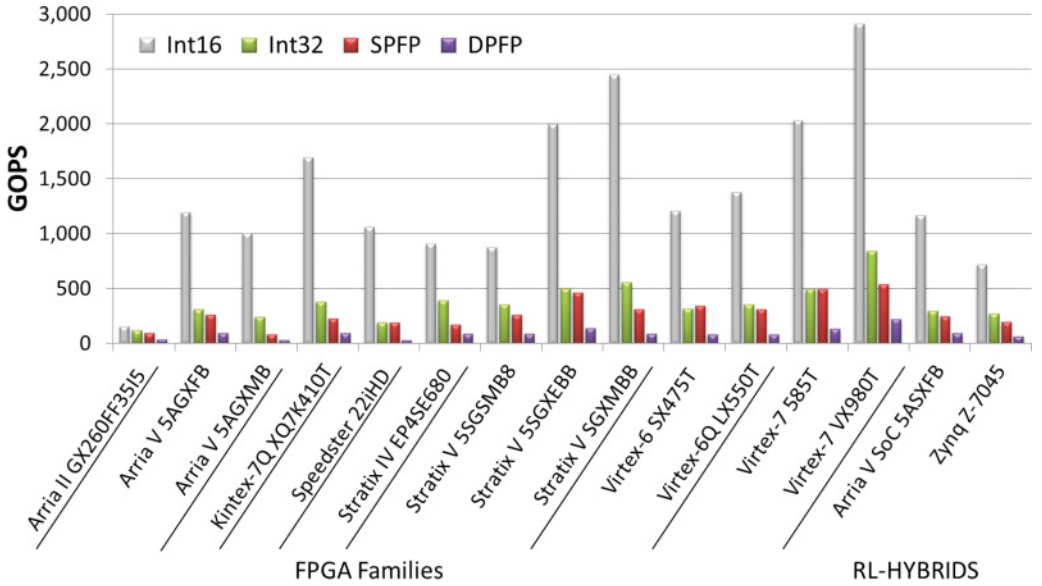


Fig. 2. CD for select reconfigurable-logic devices.

Notable results from our analysis follow. The AMD Radeon R9 295X2 had the highest SPFP score (5,745 GOPS) by a significant margin. This result stemmed from the twin GPUs on this device and was expected, given the trend of increasing computational power of GPU devices for general-purpose computing. For the Int32 and DPFP data types, the NVIDIA GTX 690 GPU (3,133 GOPS) and NVIDIA GTX Titan (785 GOPS) surpassed the R9 295X2, highlighting the different emphases between these GPU architectures. The Opteron 6380 scored highly in the Int16 precision (3,584 GOPS) even surpassing the GPUs. This result is due to both the high operating frequency (3.5 GHz) and the wider memory bus for passing data to the processing cores. GPU devices are built to specifically handle floating-point instructions and, while they are being enhanced to support other precisions, their main emphasis shows in their SPFP and DPFP scores.

In a similar manner to the fixed-logic devices, we filtered the reconfigurable-logic results into a subset of the top-scoring devices in either CD or CD/W. Additionally, we grouped the reconfigurable-logic devices into two major categories, FPGA families and reconfigurable-logic hybrid (RL-HYBRID) devices, as shown in Figure 2. The FPGA families studied span various process technologies, capacities, and architectures. The focus of this analysis is different processing architectures, and, for this study, the generational improvements in FPGA devices are used to delineate between different FPGAs. The Virtex-7 VX980T by Xilinx scores highest in all data types for the FPGA devices due to the large number of resources used for computational logic. Overall, the reconfigurable nature of the fabric in FPGAs enables them to take better advantage of their resources with smaller data types. In contrast, the SPFP results for FPGAs are significantly lower than the GPUs because SPFP functional units are much larger and require significantly more resources than the smaller fixed-point functional units. For a complete list of the FPGAs studied and their CD values, please see Table I in the appendix.

### 2.3. CD/W Results

The CD results, presented in the previous section, characterize the computational abilities of the respective devices without consideration of the power consumed. This section

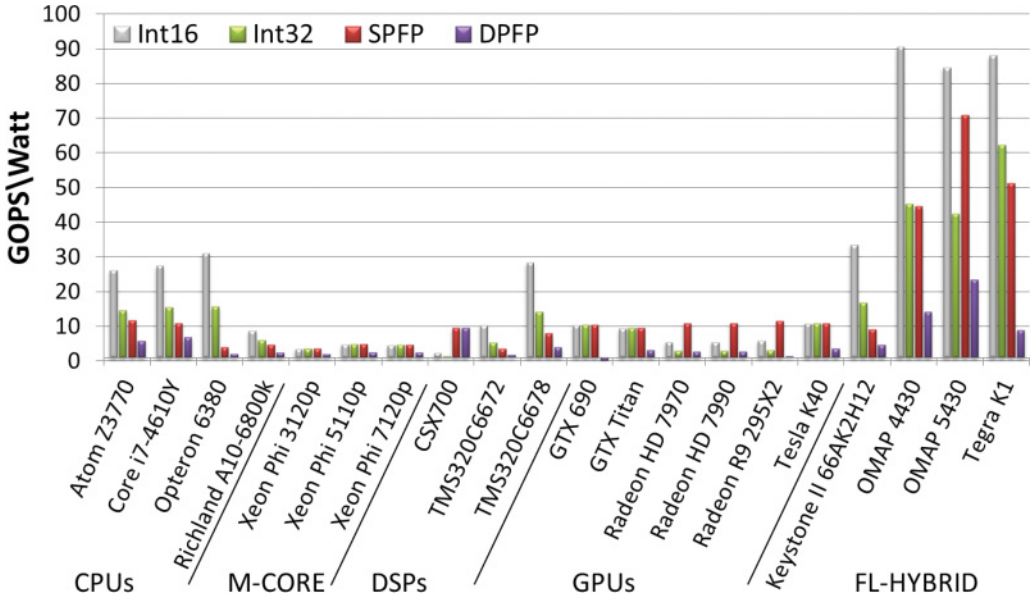


Fig. 3. CD/W for select fixed-logic devices.

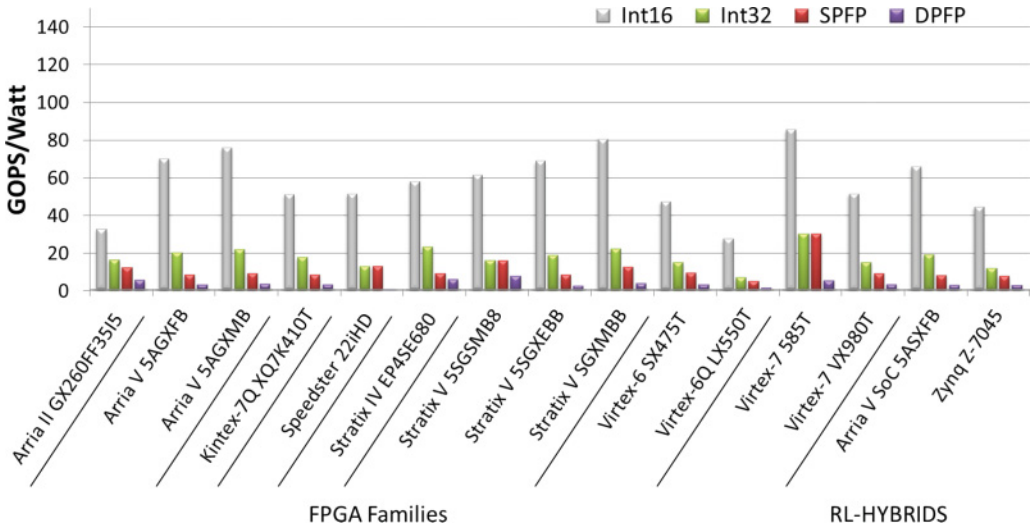


Fig. 4. CD/W for select reconfigurable-logic processors.

details how different devices excel when power is considered. The power-aware CD/W scores for fixed-logic and reconfigurable-logic processors are presented in Figures 3 and 4, respectively.

The fixed-logic CD/W data clearly show the power efficiency of the new generation of heterogeneous hybrid devices. The TI Open Multimedia Applications Platform (OMAP) 4430 hybrid SoC has the highest Int16 CD/W ratio (90.43 GOPS/W, 0.6 W) with its successor the OMAP 5430 having the best SPFP (70.92 GOPS/W, 1.37 W) and DPFP (23.39 GOPS/W, 1.37 W). Even though these devices do not have large computational resources, they achieve high CD/W due to low power consumption. Another hybrid

device, the NVIDIA Tegra K1, scores the highest in Int32 (62.24 GOPS/W, 5 W). Additionally, the data show significantly lower CD/W numbers for traditional fixed-logic processors as compared to FPGA processors and hybrid devices. This trend can be directly attributed to the large number of computational resources on many FPGAs and the higher power consumption of traditional fixed-logic devices.

For the FPGA devices, the Virtex-7 585T has the highest Int16 CD/W (85.80 GOPS/W), Int32 (30.4 GOPS/W), and SPFP (13.16 GOPS/W) due to the number of fixed and floating-point operations that can be packed on the device within its power envelope. Additionally, the Stratix V 5SGSMB8 has the highest DPFP CD/W (7.99 GOPS/W, 10.72 W). The Achronix Speedster 22iHD has the smallest process technology (22nm) of the FPGA devices, which improves its CD/W score (51.55 GOPS/W, 14.48 W), but, counterintuitively, the greater available resources on the other FPGA devices outmatch the process technology gains with higher performance per watt. Achronix employs hard-macros, sections of the fabric that are optimized for certain functions. These hard-macros are mostly focused on high-speed communication protocols and interfaces. The inclusion of these interfaces improved the I/O capability of the Achronix device but reduced its available logic area. This reduction restricts the number of computations that can be packed onto the device and therefore limits the available CD and lowers the CD/W score.

For SPFP, the CD/W results show that many FPGA processors perform on par with GPU devices. The higher power consumption of GPU devices offsets the benefits of their superior SPFP performance. For high levels of parallelism, the Radeon R9 295X2 has the highest CD/W in SPFP (11.5 GOPS/W, 500 W) of the high-powered GPU devices studied. Although high-end FPGAs in the Stratix IV and V, as well as Virtex-6 and 7, families have a much larger number of parallel operations than the Radeon R9 295X2, the achievable frequency is low compared to the GPU's operating frequency. The OMAP 5430 SoC uses very little power, which allows it to perform well in CD/W, despite having a much lower CD than other fixed-logic devices studied. For a complete list of devices with their respective CD/W values, see Table II in the appendix.

The computational device metrics provide first-order insight into the performance capabilities of devices. CD and CD/W showed the computational ability of high-performance GPUs comes at a steep power cost, and the flexibility of FPGA devices shows in small-precision performance. Hybrid devices, especially CPU/GPU hybrids, show a significant performance per watt advantage over traditional fixed-logic processors and are competing with FPGAs in power-efficient computing.

## 3. DEVICE MEMORY METRICS

To provide a thorough characterization of a computational device, we introduce two new memory metrics to quantify the ability of a processor to move information into and out of the processing cores. Building on the related research and device metrics presented in Williams et al. [2011], Sections 3.1 and 3.2 introduce the EMB and the IOB metrics. Sections 3.3 and 3.4 highlight the results of our memory metrics analysis.

### 3.1. External Memory Bandwidth

External memory bandwidth is used to describe the total bandwidth between a device and attached external memory. This metric only includes the bandwidth of usable data, excluding bits for error-correction coding. In addition, EMB does not include I/O or network-controller bandwidth, as these are typically at the cost of a user-defined interfacing implementation for an application. Although a device could access another device's memory through an I/O port, this is not considered in the calculation. Due to this assumption, on FPGA devices with user-controllable high-speed transceivers these transceivers are not included in EMB, with the exception of those used in the memory

controller implementation. The following paragraphs detail the EMB calculation for both fixed-logic and reconfigurable-logic devices.

For fixed-logic processors with built-in memory controllers, EMB is the sum of the concurrent bandwidth provided by all memory controllers. For devices that use a front-side bus, the entire bus is allocated for memory bandwidth. For reconfigurable-logic devices, the methodology employed is similar to determining CD for FPGAs. Instead of implementing computational cores, memory controllers are implemented into the logic fabric. In contrast to the CD case, limiting factors for memory controllers include the number of LUTs, ALMs/Slices, and the number of bonded IOBs.

To get a better understanding of how to calculate EMB, a step-by-step calculation for the Virtex-6 SX475T follows. A single Double Data Rate (DDR2) memory-controller IP core is instantiated on the chip, and the resource utilization is obtained from the post place-and-route report. In this example, we used the DDR2 core because it was the highest performance memory core from the vendor comparable with the device. From the place-and-route information, the most limiting resource was the number of bonded IOBs. The maximum number of DDR2 controllers that can be instantiated simultaneously is calculated by dividing the available number of bonded IOBs (840) by the number of IOBs used by a single memory controller (121). The memory-interface frequency (533MHz) is multiplied by the memory-interface width (64 bits) using the appropriate units to get the EMB of one DDR2 controller (8.528GB/s). This rate is in-turn multiplied by 6, the maximum number of DDR2 controllers instantiated, to calculate a maximum EMB of 51.168GB/s.

### 3.2. Input-Output Bandwidth

Input-output bandwidth is used to describe the total I/O capabilities of a device. Devices with dedicated ports for interfacing with memory often have additional ports for data input/output, which are not considered in the EMB calculation. Devices may also have higher bandwidth capabilities on a port that shares all or some pins with ones used for a memory interface, as in reconfigurable-logic devices. In this case, IOB would include the combination, producing the most bandwidth for the device.

There are numerous ways to characterize the I/O of a device. In single-ended I/O, one signal is made between two integrated circuits and compared to a specified voltage range or to a reference voltage. In differential signaling, two signals are made between two ICs, and the signals are compared to each other to determine the logic value [Athavale and Christensen 2005]. These two signaling methods can have differing bandwidths, even when comparing two single-ended signals to an individual differential signaling pair. When studying IOB, it is important to keep similar parameters equal when direct comparisons are desired.

IOB is calculated as the total aggregate sum of the bandwidth provided by all inputs and outputs that can operate concurrently. The highest bandwidth ports are used when there is overlap or non-concurrency. For reconfigurable-logic devices such as FPGAs, all concurrently available interfaces, including high-speed transceivers, would be counted with the exception of resources hard-coded into memory controllers. Line encoders can be used to encode data into a different format, which benefits transmission for reasons other than data throughput. Various schemes, such as 8b/10b or 64b/66b, can be employed that have varying overheads on the line rate. If an encoding scheme is used, such as 8b/10b, then the IOB represents the fraction of bandwidth that is available for real data. The aggregate sum is then added to the input/output bandwidth of any dedicated external memory controllers available on the device.

For example, consider the Nvidia Tesla C1060 GPU. There are two interfaces for data I/O on this processor, the memory interface and the PCIe bus. To compute IOB, the aggregate is taken of both interfaces, and the calculations are shown in

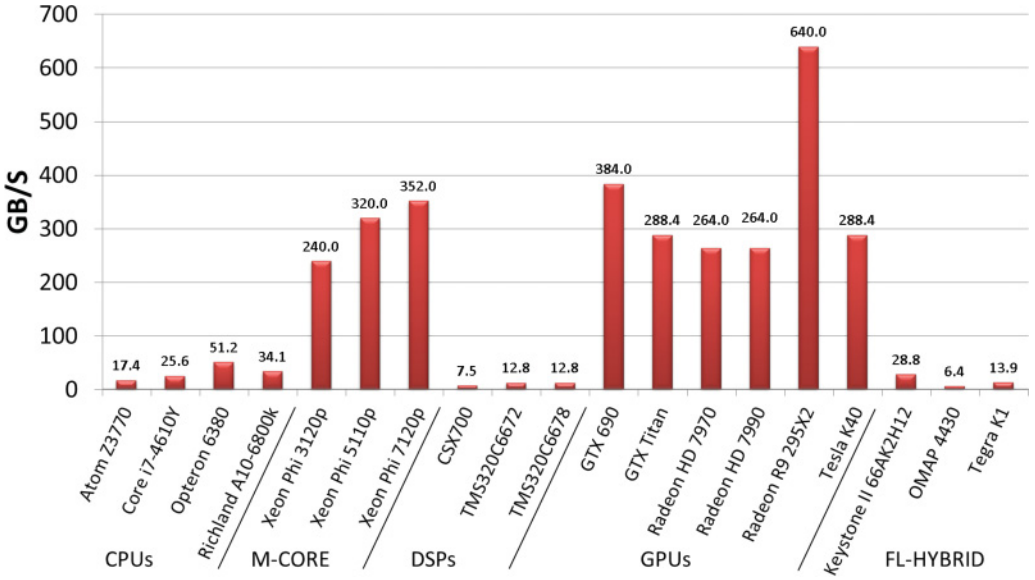


Fig. 5. EMB for select fixed-logic processors.

Equations (3)–(5). It has dedicated GDDR3 memory clocked at 800MHz on a 512-bit interface. The PCIe interface has a 500MB/s transfer rate for each lane in each of two directions,

$$IOB_{mem} = 800\,MHz \times (512\,bits/8) \times 2 = 102.4\,GB/s, \tag{3}$$

$$IOB_{PCIe} = 500\,MB/s \times 16\,Lanes \times 2 = 16\,GB/s, \tag{4}$$

$$IOB = IOB_{mem} + \times IOB_{PCIe} = 118.4\,GB/s. \tag{5}$$

### 3.3. EMB Results

This section presents highlights from the EMB analysis for the same suite of 130 fixed-logic and reconfigurable-logic devices. Figure 5 shows EMB, in GB/s, for the fixed-logic processors highlighted in the previous section. As expected, GPU devices perform best in all categories studied. GPUs are designed to handle memory-intensive applications that require large sets of streaming data. This design has fast and wide memory buses that result in high EMB. The Radeon R9 295X2 has the highest EMB (640GB/s) of the devices studied. CPU devices typically handle smaller applications using smaller sets of data; hence, they have a lesser EMB. The Intel Xeon Phi 7120p has the highest EMB (352GB/s) of the non-GPU, fixed-logic devices due to the high-speed memory interface supporting the Xeon cores on the device.

Figure 6 shows EMB for highlighted reconfigurable-logic devices and their hybrids. The Virtex-7 VX980T has the highest EMB (89.6 GB/s) due to the greater number of communication resources available as external memory controllers. These resources allow the simultaneous instantiation of more DDR2 controllers, resulting in a higher EMB score. Most of the devices require almost no logic utilization after instantiating their memory controllers, although the smallest FPGAs can use a significant number of logic resources in supporting the memory controllers. For a complete list of the EMB and IOB results for this study, please see Table III in the appendix.

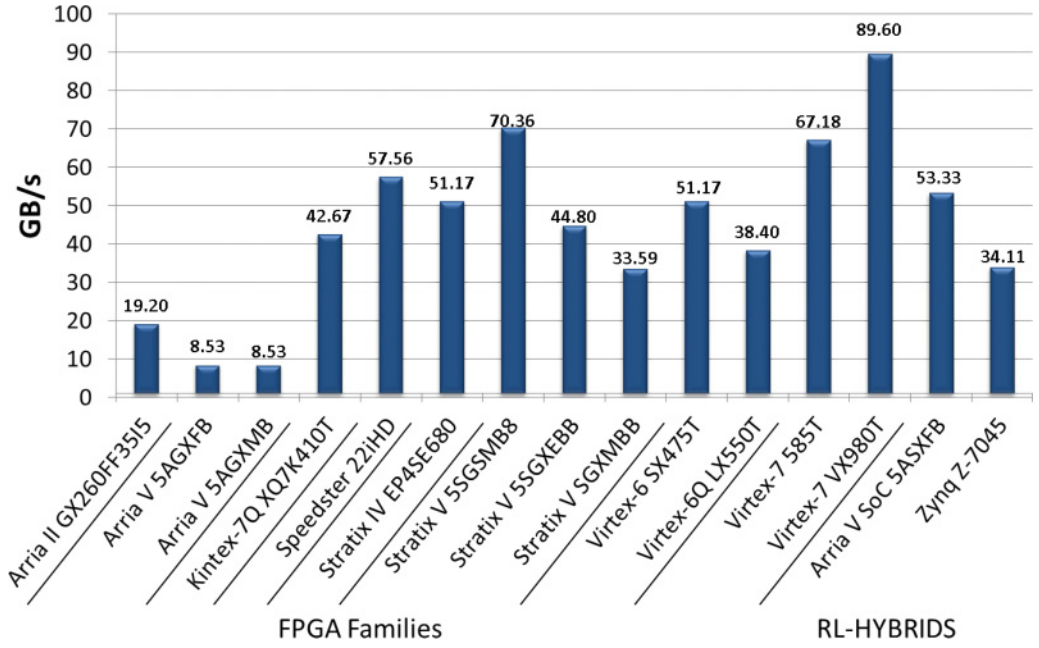


Fig. 6. EMB for select reconfigurable-logic devices.

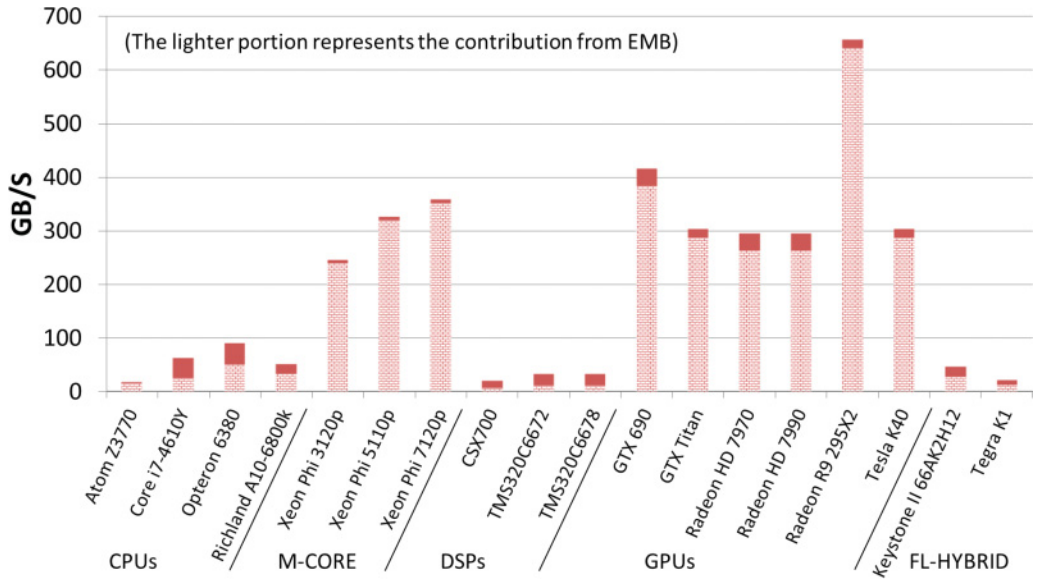


Fig. 7. IOB data for select fixed-logic devices.

### 3.4. IOB Results

The highlights from the IOB analysis present some of the differences between architectures dominated by EMB (i.e., GPUs) and more I/O diverse processing devices (i.e., FPGAs). Figure 7 shows IOB for fixed-logic devices with the lighter portion representing the contribution from EMB. This stacked approach highlights the direct

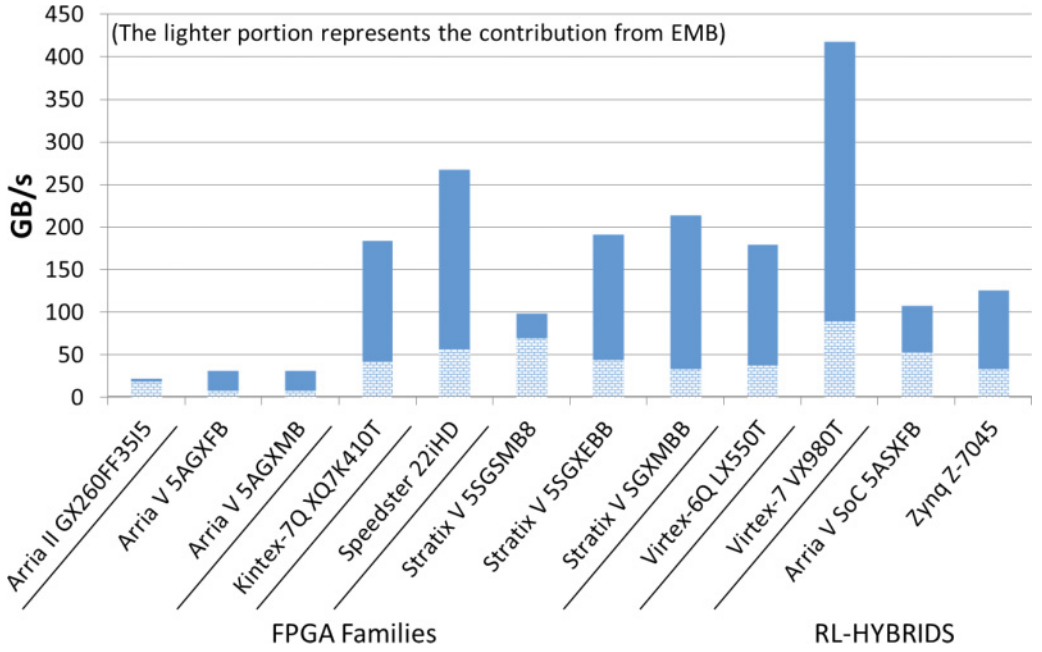


Fig. 8. IOB data for select reconfigurable-logic devices.

comparison of EMB and IOB contributions between both fixed and reconfigurable architectures. The R9 295X2 has the highest IOB (640GB/s) of the devices studied due to the high EMB scores. GPUs are optimized for 3D rendering, which requires processing on large working sets of data. Comparing against microprocessors, the amount of data that need to be processed is too large to fit in the cache of a CPU. The working sets of applications that typically run on CPUs have random memory-access patterns and are smaller than those that run on a GPU, requiring frequent fetches from off-chip memory. CPU memory interfaces have shifted from buses to a very fast group of serial data lines communicating via packets with much lower latency, such as AMD's HyperTransport or Intel's Quick Path Interconnect (QPI). This trend means that, as CPU core counts and IOB increase, streaming applications can be more effectively parallelized on them.

For reconfigurable-logic devices, shown in Figure 8, the IOB tends to be made up of different I/O connections. The Virtex-7 VX980T has the highest IOB (417.43GB/s) of FPGA devices due to the large EMB and the high availability of I/O resources, specifically bonded IOBs. The Speedster 22HiD uses hard macro blocks for I/O, which reduce logic available for computation but help the device score well in IOB (267.41GB/s) versus other FPGA devices. Smaller FPGAs struggle in IOB due to the low number of pins available for I/O operations and a lower number of communication transceivers.

The memory metrics provide first-order insight into the data movement capabilities of a device. EMB and IOB showed us that the high memory-metric scores of new GPU devices are mostly due to their high-speed memory controller interfaces. In contrast, reconfigurable-logic devices have greater focus on providing communication flexibility in more ways than simple memory controller interfaces, hence better IOB scores. These insights can help application and hardware designers tailor their device choices based on the types of data being manipulated in their desired applications.

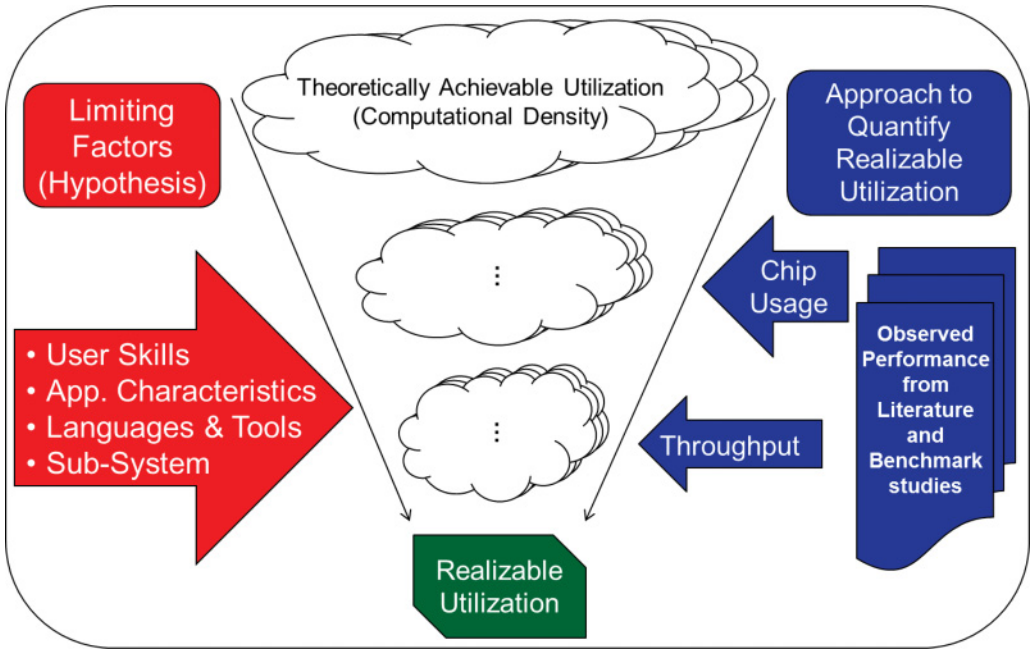


Fig. 9. Concept diagram of realizable utilization.

## 4. REALIZABLE UTILIZATION

Metrics provide a means to facilitate objective device comparison, tradeoff analysis, and first-order performance prediction. This approach provides an upper bound of a device's capability. The fraction of a device's capability that can be utilized in a particular application cannot be determined without application performance data. To address this issue, we introduce the concept of RU, which quantifies the fraction of CD that an application achieves within a specific implementation. The RU metric provides insight on device to application mapping in terms of achieved performance, tempering the computational metrics with realistic expectations. In this section, computational density is contrasted with performance data from both literature and benchmarking results. Additionally, to demonstrate how RU can showcase the tradeoffs in performance versus portability, code with intrinsic functions is compared with optimized libraries and benchmarks coded for portability.

### 4.1. RU Methodology

There are many factors that can reduce the performance of a device, including application characteristics, tools, and user experience. Realizable utilization is a method to quantify the difference between a device's theoretical performance and the actual performance a user can expect to achieve. Since benchmarking every device with every application is impractical, RU allows developers to estimate their application's projected performance on a particular device. A device performance metric, such as CD, scaled using RU data from published technical literature and benchmarking, can be used to estimate the realized performance expected in a specific application.

In Figure 9, illustrating the RU methodology, the theoretical computational capacity, represented by CD, is reduced by various factors such as developer experience, application characteristics, tools used, and so on. We observed this throttling trend through data found in technical publications and benchmarking studies. RU starts with CD

representing the theoretical computational capacity of a device. Performance data are then collected from either scholarly publications or benchmarking experience. The application throughput from technical data and benchmarks shows the performance achieved for a specific platform, application, and implementation. This performance data are used to compare observed throughput with CD, yielding the RU score. The RU score becomes more applicable as the amount of literature increases; for new technologies, these data may be sparse. Benchmarking requires more development time and effort but, since it uses more hardware resources and is closer to the desired application, it provides more accurate data. Using data acquired from benchmarking provides more revealing RU scores, since the developer frequently tunes the application to the hardware or, conversely, tunes the hardware to the application, as in FPGAs.

### 4.2. Calculating RU

Once CD is determined, the RU metric is calculated by dividing the observed throughput ($T$) in OPS by the CD of the device used in the application, as seen in Equation (6). The device's CD is multiplied by a scaling factor representing the fraction of the device used by the application. This factor, $\alpha$, depends on the implementation of the application. For example, if an application only uses one core of a quad-core CPU, then the factor $\alpha$ is 0.25. The factor $\alpha$ is necessary because some applications have not been parallelized and, without adjusting the available CD, the comparison between applications would not be as insightful,

$$RU = \frac{T}{\alpha \cdot CD_{device}}. \tag{6}$$

The developer's knowledge of their application and its implementation allows $\alpha$ to best be calculated during benchmarking. When the information found in publication sources does not provide enough data to reliably determine $\alpha$, then a ratio of 1 is assumed. This assumption is based on the hypothesis that most developers who are publishing their work and having it peer reviewed will be trying to maximize performance of their application. If the application is not using all of the main resources of the device, then the developer generally includes enough information to calculate $\alpha$.

Since the CD value represents the theoretical maximum throughput, Equation (6) shows that the RU metric is bounded below by zero and above by 1. While RU is a ratio, it is expressed as a percentage. From this alternate perspective, RU is the percentage of a device's theoretical performance achieved by an application. This analysis provides insight for developers, not only before coding their application but also during the development cycle.

### 4.3. Using RU

Once the RU metric has been calculated, it provides useful insight for various program types. One use of RU is in the device design process. Device architects can use RU when developing novel devices by comparing a new architecture to similarly structured existing devices. The RU score indicates what program areas are most likely to map well on a future device and that information can be incorporated into the device development process. For example, a novel many-core device could be compared with existing RU scores for GPUs, showing similar device enhancements for data movement could help improve the performance of dense linear algebra.

Another use of RU, from a system designer's standpoint, is to assist in selection of an appropriate acceleration platform before significant costs are expended on cutting-edge hardware. Applications with similar structure or kernels can be analyzed to see what platforms are making the most of the available resources. For example, in a system that is being built to run a significant number of FFTs, the RU scores can show that a DSP or FPGA option might be the most effective use of resources. This insight mitigates

some of the risk with developing applications on new platforms and allows developers to narrow the field of application accelerators.

Finally, software developers can use RU to gain feedback while developing their applications. During the optimization cycle of development, it can be difficult to judge when the maximized performance from the optimization has been reached and how much more optimization performance can be expected. RU allows developers to compare the kernels or applications that are undergoing optimization, such as SIMD optimization with ARM NEON acceleration, to similar applications and kernels. This comparison can help the developer judge how much more performance they can expect to achieve from their application and then decide if additional performance is worth the time and cost.

### 4.4. Arithmetic Kernels for RU

The authors investigated the application of RU through a literature study of three arithmetic-heavy application kernels. The first kernel in this study, matrix multiplication (MM), includes both simple matrix-multiplication kernels, Equation (7), and generic matrix-multiplication (GEMM) kernels, Equation (8).

$$(\mathbf{AB})_{ij} \quad = \quad \sum_{k=1}^{m} \mathbf{A}_{ik}\mathbf{B}_{kj}. \tag{7}$$

GEMM kernels are common subroutines used as part of the Basic Linear Algebra System (BLAS). For Equation (8), the values $\alpha$ and $\beta$ are scalar coefficients,

$$\mathbf{C} \quad \leftarrow \quad \alpha\mathbf{AB} \quad + \quad \beta\mathbf{C}. \tag{8}$$

The second kernel type studied is matrix decomposition (MD) including Cholesky, Lower Upper (LU), and QR decompositions. Cholesky decompositions, Equation (9), break down a real, positive-definite matrix ($\mathbf{A}$) into an upper triangular matrix ($\mathbf{R}$) with positive diagonal coefficients and the Cholesky factor ($\mathbf{R}^T$) [Dongarra et al. 1979],

$$\mathbf{A} \quad = \quad \mathbf{R}^T\mathbf{R}. \tag{9}$$

LU decompositions, Equation (10), reduce a square matrix ($\mathbf{A}$) into a lower triangular matrix ($\mathbf{L}$) and a upper triangular matrix ($\mathbf{U}$) [Lancaster and Tismenetsky 1985],

$$\mathbf{A} \quad = \quad \mathbf{LU}. \tag{10}$$

The final type of matrix decomposition, QR, is used for Eigenvalue calculations in many applications. Equation (11) shows how QR decomposition breaks the square matrix ($\mathbf{A}$) into an orthogonal matrix ($\mathbf{Q}$) and a upper triangular matrix ($\mathbf{R}$) [Bhaskar 2006]. In contrast to the previous LU decomposition, the QR decomposition always exists,

$$\mathbf{A} \quad = \quad \mathbf{QR}. \tag{11}$$

The third kernel area studied in this article is n-body simulations. Two major types of n-body simulations are molecular dynamics simulations and astronomical or gravitational simulations. Molecular dynamics simulations compute the electrostatic forces and interactions between molecules within the simulation [Bailey 1995], and gravitational simulations compute the gravitational forces from a set of spatially related objects [Aarseth and Aarseth 2003].

### 4.5. Literature Study Results

To build on the metrics results, the authors selected GPU-based devices for more in-depth literature study. This literature study was conducted to highlight the use of RU

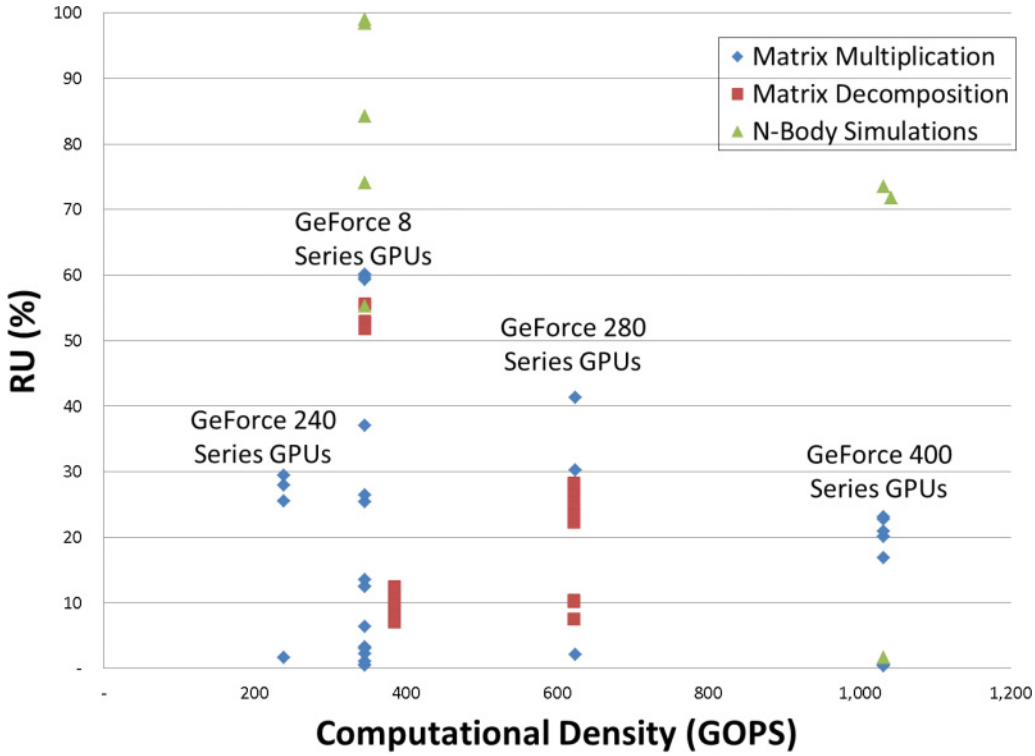


Fig. 10. Combined realizable utilization results.

without investing the time and money that benchmarking requires. Surveying both academic and vendor articles, we analyzed the most common GPU architectures by calculating the achieved throughput and using that information to calculate RU. These results are separated into three key kernel types: matrix multiplication (Table IV in the appendix), matrix decomposition (Table V), and n-body simulations (Table VI). Figure 10 shows all the RU scores plotted together for each kernel type.

For the matrix-multiplication kernels, the best RU scores for GPU devices are found in the GeForce 8 Series GPUs. Many of the various optimizations, such as CUDABLAS, generated groupings of similar performance marks within an architectural family. These groupings highlighted key optimization levels by clustering similar optimization scores close to each other in each family column. Of the three kernels studied, matrix multiplication was the most common and, due to the well-known features of its operations, matrix multiplication provides significant RU results. The peak matrix-multiplication RU scores highlight an obvious trend, significantly decreasing RU as the CD increases. This trend implies that, in most cases, raw performance is still increasing with more powerful chips, but the realized performance is not keeping pace with the theoretical capacities of the devices. This decreasing RU trend points to applications and/or tools as the limiting factor in achieved performance.

The highest-scoring GPU device in the second kernel area, matrix decomposition, is the GeForce 8800 GTX in the 8-series family, with an RU score of 55.56%. This device has the same basic architecture as the Tesla C870 of the same family, the highest scoring GPU in matrix multiplication, and is one of the most commonly found devices in this literature study.

The matrix-decomposition RU scores further reveal the trend of decreasing RU scores with increasing device capacity and show similar performance patterns as matrix multiplication.

The final kernel area in this study is n-body simulations. Both molecular dynamics simulations, as well as astrophysical simulations, are included in the data set plotted in Figure 10. The data in n-body simulations shows the highest range of scores (1% to 99%) and includes the highest scores overall. These high scores point to a better fit between the application's structure and the GPU device's hardware resources. The iterative and parallel nature of the n-body style of application fits well into the streaming processor model employed by GPUs. While the devices with higher CD scored significantly better in RU here than they did in other application areas, they still fell behind the older GeForce 8-series GPUs.

The data points reinforce the significant prevalence of the GeForce 8 series. These older GPUs are more cost-efficient to obtain and employ the same device architecture. Combined, these three sets of results show that the higher CD devices tend to have lower RU scores, especially in matrix-based kernels. While overall raw performance is increasing as the device CD grows, the downward trend in RU shows that applications and tools are not yet able to capitalize fully on the added computational resources. This trend could be caused by many different issues, including device tools, developer experience, application characteristics, architecture bottlenecks, and others. Determining which of these issues are most responsible for reducing RU is being considered for future work, but one example might include memory bandwidth utilization. This utilization would be linked to the level of library or implementation optimization for the newer GPU devices. As tools and algorithm implementations improve, issues such as utilization of memory bandwidth may be improved to dramatically improve the RU scores. One of the major issues with the literature-based approach is that up-to-date results are hard to find given the time for academic optimization and publication. One way to address this issue is to do actual benchmarking. The following section highlights some of our results in this area.

### 4.6. RU Benchmarking Results

Expanding on the previous literature study, benchmarking follows as the next step in using RU. For this initial exploration, multiple devices were compared based on several computational kernels and library implementations. Using vendor-optimized libraries [NVidia 2015; Intel 2014; Whaley et al. 2001], hand-optimized kernels, and codes optimized for portability, we benchmarked a combination of nine kernels on 11 devices with a total of 29 individual data points. Kernels included several common types found in compute-heavy applications (Matrix Multiplication (MM), Fast Fourier Transform (FFT), Single Value Decomposition (SVD), etc.). Advanced libraries, ATLAS, for example, were tested against hand-optimized algorithms and code optimized for portability. RU allows the quantification of the performance difference between implementation types with respect to device metrics like CD. Figure 11 shows, in decreasing order of RU, the results of this study.

The first key insight from Figure 11 is the clear performance gained using mature libraries on high-performance processors. The highest RU score (91.4%) was found on a Tesla K20X using a matrix-multiplication algorithm based on the high-performance cuBLAS library. This library from NVidia provides key BLAS support for their GPU architectures. When the applications in question, such as large matrix operations, can saturate all the GPU resources with limited data dependency, we see very high RU scores. If the kernel in question is more memory-intensive or has complex data dependencies, such as FFTs, then the RU of the GPUs tends to be lower. These results yield a wide variance in the RU scores based on kernel structure.

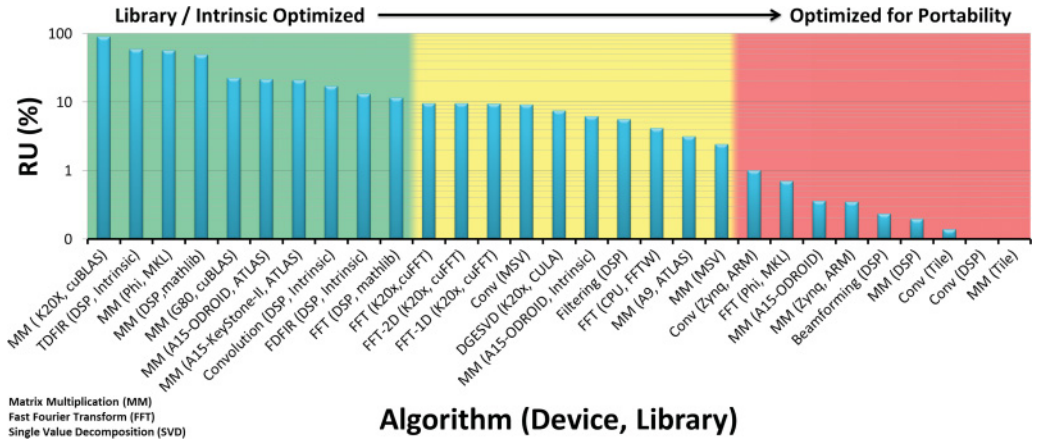


Fig. 11. Realizable utilization benchmarking results.

Another insight is the excellent RU gains from using SIMD processing when the algorithm supports it. The time domain finite impulse response (TDFIR) filter, coded with DSP intrinsic functions, achieved the second highest RU score (59.7%) on the TI KeyStone-I DSP device. In contrast to the high performance of tuned libraries, code that was optimized for portability between platforms sacrificed a significant amount of RU to achieve their general applicability. By limiting to generic coding techniques and instructions that could be easily carried between different devices, a major tradeoff between portability and performance is illustrated. For example, this tradeoff is shown by the two matrix-multiplication implementations on the TI DSP that range from 50% RU when using a matrix-operation library to 0.2% using generic instructions.

Between the high-performance libraries and the lower-performance codes optimized for portability is a middle ground, composed of an interesting set of application/device combinations. Counterintuitively, even the highly optimized libraries frequently score mid-range (e.g., 11.7%) on FFT algorithms. This result indicates that the structure of FFTs does not map as well to these devices as linear-algebra kernels. Therefore, alternative device architectures may be of interest for FFT-heavy applications. Table VII in the appendix shows the numerical RU scores for the devices included in this case study.

From our initial explorations into applying RU to FPGAs, we have seen promising RU scores for computationally dense algorithms. For example, preliminary studies are showing RU scores ranging from approximately 75 to 94% for dense applications such as the Smith-Waterman bio-informatics algorithm. Building on the existing work, FPGA devices are being explored but their device complexities are extending beyond the scope of this study. Subsequent publications are planned to expand on the details of RU's application to FPGAs.

This section introduced the realizable utilization metric to quantify the fraction of the CD that applications achieve. RU from literature studies showed the strengths of mature architectures and weakness of less-developed devices by showcasing the differences in achieved utilization. From our benchmarking results, RU showed significant utilization gains with intrinsic optimizations and highly optimized libraries in contrast to code optimized for portability. Together, these results highlight how the RU metric can be used to help developers predict application performance on various computational devices.

## 5. CONCLUSIONS

In this article, the authors have used computational-density metrics to survey a new generation of various device architectures. Also, the metrics methodology was expanded to include new memory-based metrics to characterize the data movement abilities of processors. Further, a new realizable utilization metric was introduced to quantify the theoretical versus achieved application performance.

First, this work evaluated the computational-density data of a new generation of devices, clearly showing the strength of new hybrid devices in terms of computation per watt. The hybrid OMAP processors score the highest CD/W in all precisions due to their small power envelope. Additionally, for FPGAs, the Virtex-7 585T FPGA device shows the highest CD/W with fixed precisions because of its power-efficient reconfigurable resources. Another insight observed in Section 2 was that the Radeon R9 295X2 GPU has a distinct advantage in SPFP calculations when power is not considered due to the high clock rates of the shader units and the sheer core count.

Second, this article enhanced our existing methodology [Williams et al. 2011] for device metrics by introducing new memory metrics, EMB and IOB, and used these metrics to discuss the off-chip memory bandwidth of devices. The EMB and IOB results, in Section 3, show that GPUs, the Radeon R9 295X2, for example, have the highest external bandwidth, with wide memory controllers working at very high frequencies. FPGAs emphasize data connectivity, with high IOB, but their lower operating frequency keeps them from matching GPU memory bandwidth. Newer CPUs, particularly the Xeon E5-2670, can achieve higher IOB compared to earlier CPUs due to the shift from the front-side bus connections to higher speed interfaces such as QPI.

Finally, realizable utilization is a new metric introduced in Section 4 and is used to temper computational density with real application performance. From the GPU literature study, we noticed a significant downward RU trend with matrix-based applications as the architecture generations became newer. Raw GPU performance continued to increase, but the computational capacity outpaces the application performance. The benchmarking analysis showed that RU can be used to demonstrate and quantify the tradeoff between highly optimized code with low portability and general code with higher portability. The analysis showed the best RU scores occur with highly tuned libraries such as cuBLAS or optimized SIMD instructions and dense computational loads. When coded for portability, using generic structures, similar applications sacrifice a significant amount of performance for compatibility. Overall, the metrics presented in this article provide a first-order analysis of device characterization and insight into the strengths of a wide variety of device architectures.

Future work is planned to allow for user-defined parameters when calculating certain metrics. These parameters include the addition of more operations and enhanced application-to-metric mapping using automated tools. Additionally, the RU work is being expanded in terms of volume and type of devices studied, including FPGAs, to allow a statistical upper bound, such as a confidence interval, to be more useful in predicting RU. This expansion will allow users to more accurately determine which of many devices would best fit their algorithm. Our ultimate goal is to allow users to define and customize the individual attributes used to calculate each metric to best suit their application.

## ELECTRONIC APPENDIX

This appendix has been posted online for space considerations. Please find the complete appendix at http://www.chrec.org/pubs/JR-TRETS-2015-Appendix.pdf

## ACKNOWLEDGMENTS

The authors thank Altera, Xilinx, and ARM for their tool and hardware donations and Andrew Milluzzi, Nick Wulf, Tyler Lovelly, Steven Figulin, Aishwarya Dhandapani, and Ishan Dalal for helping collect metrics data and benchmarks.

## REFERENCES

S. Aarseth and S. J. Aarseth. 2003. *Gravitational N-Body Simulations*. Cambridge University Press. http://books.google.com/books?id=Xo8eaQzs0YoC.

A. Athavale and C. Christensen. 2005. *High-Speed Serial I/O Made Simple, A Designers' Guide, with FPGA Applications*. Xilinx Connectivity Solutions.

D. H. Bailey. 1995. *Proceedings of the Seventh Siam Conference on Parallel Processing for Scientific Computing*. Siam. http://books.google.com/books?id=FgDYbavV-R4C.

Sergio Barrachina, Maribel Castillo, Francisco Igual, Rafael Mayo, and Enrique Quintana-Ort. 2008. Solving dense linear systems on graphics processors. In *Euro-Par 2008 Parallel Processing*, Emilio Luque, T. Margalef, and Domingo Bentez (Eds.). Lecture Notes in Computer Science, Vol. 5168. Springer, Berlin, 739–748. http://dx.doi.org/10.1007/978-3-540-85451-7_79

Sergio Barrachina, Maribel Castillo, Francisco D. Igual, Rafael Mayo, Enrique S. Quintana-Orti, and Gregorio Quintana-Orti. June 7, 2009. Exploiting the capabilities of modern GPUs for dense matrix computations. *Concurrency and Computation: Practice and Experience* (June 7, 2009).

Bathan Bell and Michael Garland. 2009. Implementing sparse matrix-vector multiplication on throughput-oriented processors. *High Performance Computing, Networking, Storage and Analysis, 2009. SC 2009. International Conference for* (November 2009).

Robert G. Belleman, Jeroen Bedorf, and Simon F. Portegies Zwart. 2008. High performance direct gravitational n-body simulations on graphics processing units II: An implementation in CUDA. *New Astron.* 13 (Feb. 2008). Issue 2.

Bhaskar. 2006. *Applied Mathematical Methods*. Pearson Education. http://books.google.com/books?id=D4DA7rWWWPYC.

Doug Burger, James R. Goodman, and Alain Kägi. 1996. Memory bandwidth limitations of future microprocessors. In *ISCA'96: Proceedings of the 23rd Annual International Symposium on Computer Architecture*. ACM, New York, NY, 78–89. DOI:http://dx.doi.org/10.1145/232973.232983

Jose M. Cecilia. The GPU on the Matrix-Matrix Multiply: Performance Study and Contributions.

Andre DeHon. 1996. *Reconfigurable Architectures for General-Purpose Computing*. Technical Report. Massachusetts Institute of Technology, Cambridge, MA, USA.

J. J. Dongarra, Society for Industrial, and Applied Mathematics. 1979. *Linpack Users' Guide*. Society for Industrial and Applied Mathematics. http://books.google.com/books?id=AmSm1n3Vw0cC.

Tsuyoshi Hamada and Toshiaki Iitaka. March 5, 2007. The chamomile scheme: An optimized algorithm for n-body simulations on programmable graphics processing units. *NewAstron.* (March 5, 2007).

Intel. 2014. Intel math kernel library reference manual. 072, MKL 11.2 (2014). https://software.intel.com/en-us/mkl_11.2_ref_pdf.

P. Lancaster and M. Tismenetsky. 1985. *The Theory of Matrices: With Applications*. Academic Press. http://books.google.com/books?id=m8z6Xh1A3t8C.

Andrew Milluzzi, Justin Richardson, Alan George, and Herman Lam. 2014. A multi-tiered optimization framework for heterogeneous computing. *IEEE Proc. of High-Performance Extreme Computing Conference* (September 2014).

NVidia. 2010a. NVidia SDK Core. http://developer.nvidia.com/cuda-toolkit. (2010).

NVidia. 2010b. NVidia SDK DirectCompute Core. http://developer.nvidia.com/cuda-toolkit. (2010).

NVidia. 2015. CUBLAS LIBRARY. 7.0 (2015). http://docs.nvidia.com/cuda/pdf/CUBLAS_Library.pdf.

Lars Nyland, Mark Harris, and Jan Prins. 2007. Fast n-body simulation with CUDA. *GPU Gems 3* (2007).

J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. 2008. GPU computing. *Proc. IEEE* 96, 5 (may 2008), 879–899. DOI:http://dx.doi.org/10.1109/JPROC.2008.917757

Shane Ryoo, Christopher I. Rodrigues, Sara S. Baghsorkhi, Sam S. Stone, David B. Kirk, and Wen-mei W. Hwu. 2008. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'08)*. ACM, New York, NY, 73–82. DOI:http://dx.doi.org/10.1145/1345206.1345220

Ashley Saulsbury, Fong Pong, and Andreas Nowatzyk. 1996. Missing the memory wall: The case for processor/memory integration. In *ISCA'96: Proceedings of the 23rd Annual International Symposium on Computer Architecture*. ACM, New York, NY, 90–101. DOI:http://dx.doi.org/10.1145/232973.232984

Gurindar S. Sohi and Manoj Franklin. 1991. High-bandwidth data memory systems for superscalar processors. *SIGOPS Operat. Syst. Rev.* 25, Special Issue (1991), 53–62. DOI:http://dx.doi.org/10.1145/106974.106980

Vasily Volkov and James Demmel. 2008a. LU, QR and Cholesky Factorizations using Vector Capabilities of GPUs. http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-49.html. (May 2008).

Vasily Volkov and James W. Demmel. Nov. 15–21, 2008b. Benchmarking GPUs to tune dense linear algebra. *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for* (Nov. 15–21, 2008).

R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. 2001. Automated empirical optimization of software and the ATLAS project. *Parallel Comput.* 27, 1–2 (2001), 3–35. Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 (www.netlib.org/lapack/lawns/lawn147.ps).

J. Williams, A. George, J. Richardson, K. Gosrani, C. Massie, and H. Lam. 2011. Characterization of fixed and reconfigurable multi-core devices for application acceleration. *ACM Trans. Reconfig. Technol. Syst.* 3, 4 (2011), 19:1–19:29.

J. Williams, A. George, J. Richardson, K. Gosrani, and S. Suresh. July 7–10, 2008a. Computational density of fixed and reconfigurable multi-core devices for application acceleration. *Proc. of Reconfigurable Systems Summer Institute 2008 (RSSI)* (July 7–10, 2008).

J. Williams, A. George, J. Richardson, K. Gosrani, and S. Suresh. Sep. 23–25, 2008b. Fixed and reconfigurable multi-core device characterization for HPEC. *Proc. of High-Performance Embedded Computing Workshop (HPEC)* (Sep. 23–25, 2008).

Received March 2015; revised January 2016; accepted January 2016