In this chapter, we put together the sizings from Chapter 4 and 7, the growth rates from Chapter 7, and the instruction requirements from Chapter 8 to form a unified area model for *RP*-space, a large class of reconfigurable processing architectures. The area model gives us a first order size estimate for reconfigurable computing devices based on the key parameters identified in the previous chapters. We use this model to estimate peak computational density as a function of granularity and on-chip instruction store sizes. We also use it to characterize the way computational efficiency decreases as application granularity and path lengths differ from the architecture's optimal points.

9.1 Model and Assumptions

We assume an array of homogeneous, general-purpose processing elements. For pedagogical purposes, no special-purpose processing units are included. The area for each bit processing element is taken to include:

- Fixed area for the computational function
- Amortized storage space for instructions
- Storage space for data
- Space for interconnect resources
- Amortized space for control

We compute the area per bit processing element as:

$$\begin{aligned} A_{bit\_elm} &= A_{fixed} + \underbrace{N_{SW}(N_p, w) \cdot A_{SW}}_{\text{interconnect}} + \underbrace{\left(\frac{c}{w}\right) \cdot n_{ibits} \cdot A_{mem\_cell}}_{\text{instruction memory}} + \\ & \qquad \underbrace{d \cdot A_{mem\_cell}}_{\text{data memory}} + \underbrace{\left(\frac{N_{ctrl}}{N_p}\right) \cdot A_{ctrl}(c_{total})}_{\text{control area}} \end{aligned} \tag{9.1}$$

Table 9.1 summarizes the parameters used in Equation 9.1.

$A_{mem\_cell} = 1200\lambda^2$ is typical of static memory, which we will assume here. Memory cells packed into large arrays are likely to be denser, on average, than small arrays or isolated memory cells. Dynamic memory cells may be a factor of four smaller in large arrays, where appropriate.

Equation 9.1 assumes that interconnect area is proportional to the number of switches. In Sections 7.6 and 7.9, we saw that switch growth rates match or determine interconnect growth rate. In Section 7.9, we did see that wiring might dominate switch growth for large w, which is not accounted by Equation 9.1. $A_{SW} = 2500\lambda^2$ is a constant of proportionality intended to match the number of switches to the empirical interconnect areas typically seen rather than a model of any particular interconnect geometry. Table 9.2 summarizes the number of switches as a function of

Parameter	Role	Assumed Value
A_{bit_elm}	Area per bit processing element	
A_{fixed}	Fixed area per compute element (LUT mux, output flip-flop, buffers)	20Kλ^2
w	Datapath width – number of bit elements controlled by one instruction	
c	Contexts – number of instructions stored per group of w processing elements	
c_{total}	Total number of instruction or data contexts addressed by controller	
n_{ibits}	Number of bits in each instruction	64
A_{mem_cell}	Area of a configuration or data memory cell	1200λ^2
N_p	Number of bit processing elements in the array	
$N_{SW}(N_p, w)$	Number of switches per bit processing element	[Eq. 7.24]
n	Tree arity in modeled hierarchical interconnect	2
k	Number of LUT inputs	4
p	Rent parameter for network	0.5
A_{SW}	Amortized area of each switch	2500λ^2
d	Number of data bits per bit processing element	
N_{ctrl}	Number of independent stream controllers	
$A_{ctrl}(c_{total})$	Area of instruction stream controller	0.3M$\lambda^2 \cdot \log_2(c_{total})$

Table 9.1: Summary of Area Model Parameters

N_p and w for $p = 0.5$, as will be used here. This is the same data which was plotted in Figure 7.5; for $p = 0.5$, the only difference is that we use $\frac{N_p}{w}$ as the network size when determining N_{SW} (See Equation 7.24).

For devices with multiple contexts, a controller manages the selection and sequencing of instructions in the array. The area we use for A_{ctrl} is a rough estimate based on a sampling of processor implementations (See Table 9.3). We assume that the area in the controller is proportional to the number of instruction address bits, $\log_2(c_{total})$. FPGAs traditionally have a single context, making $N_{ctrl} = 0$, while processors have controllers composing the program counter and branching logic.

FPGA Example Traditional FPGAs have $w = 1$ and $c = 1$. Equation 9.1, for $N_p = 4096$, computes $A_{bit\_elm} \approx 870\text{K}\lambda^2$. Comparing with Table 7.1, we see this is in the range of conventional devices.

PADDI-2 Example PADDI-2 is made from 48, 16-bit units. Each has an 8 instruction memory ($c = c_{total} = 8$) and effectively 6 words of data per compute element, $d = 6$. PADDI-2 has

$\frac{N_p}{w}$	N_{SW}	$\frac{N_p}{w}$	N_{SW}	$\frac{N_p}{w}$	N_{SW}
1	0	32	100	1024	252
2	16	64	131	2048	281
4	31	128	162	4096	311
8	49	256	192	8192	340
16	69	512	222	16384	370

Table 9.2: $N_{SW}(N_p, w)$ for $p = 0.5$, $k = 4$, $n = 2$

Design	**Controller Area**
MIPS-X [HHC+87, Cho89]	8Mλ^2
PA-RISC [YFJ+87]	12Mλ^2
VIPER [GNAB93]	12Mλ^2

Table 9.3: Area for Instruction Control Sampling

3-inputs per EXU, $k = 3$, and an initial convergence of $n = 4$. Equation 9.1 predicts 370Kλ^2 per bit operation or 284Mλ^2 for the entire array, which is about half the size of the prototype PADDI-2 die which is 576Mλ^2.

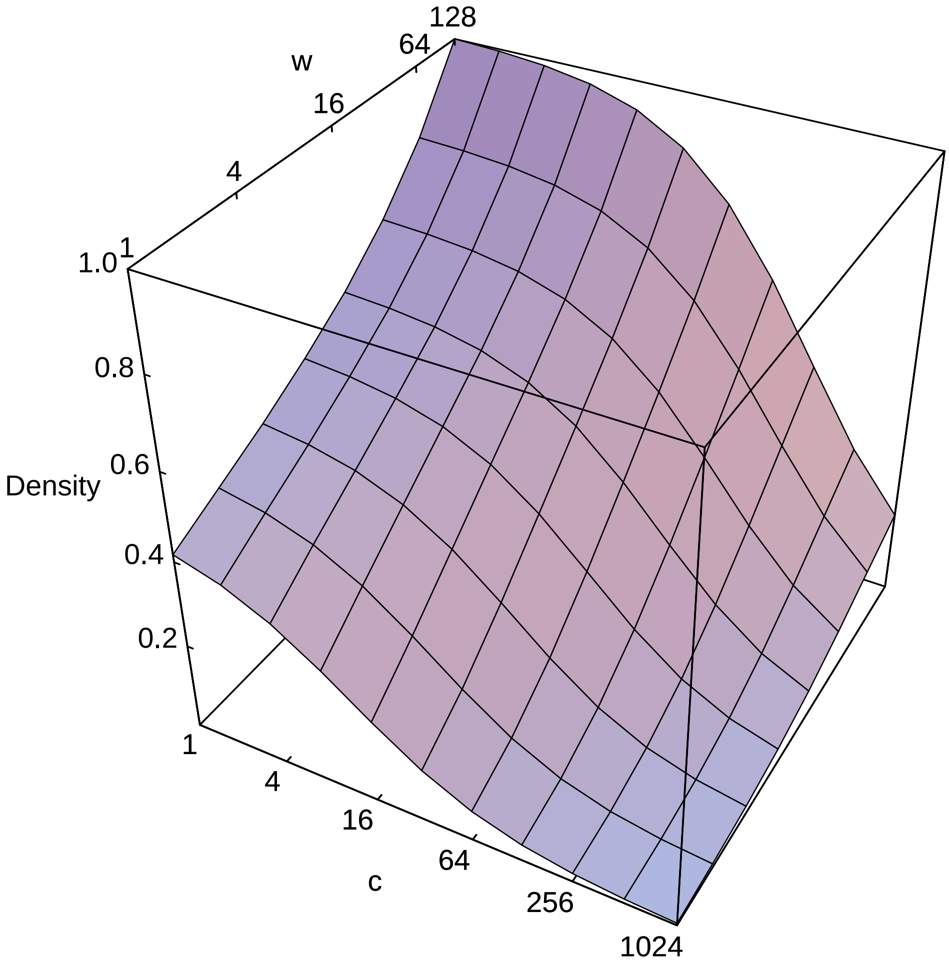
9.2 Peak Performance Density

Using the model, we can examine the peak computational densities from various architectural configurations in *RP*-space. Figure 9.1 plots computational density against datapath width, w, and the number of instructions per function group, c. As w increases there is more sharing of instruction memories and less switches required in the interconnect resulting in smaller bit processing element cell sizes or higher densities. As c increases, there are more instructions per compute element resulting in lower densities. The effect of more instructions is more severe for smaller datapath widths, w, since there are less processing elements against which to amortize instruction overhead.

For single context designs, there is only a factor of 2.5$\times$ difference in density between single bit granularity and 128-bit granularity. At this size, network effects dominate instruction effects, and the factor of difference comes almost entirely from the difference in switching requirements. For heavily multicontext devices at the same number of instruction contexts, the difference between fine and coarse granularity is greater since the instruction memory area dominates (See also Figure 9.2). At 1024 contexts, the 128 bit datapath is 36$\times$ denser than an array with bit-level granularity.

As the number of contexts, c, increase, the device is supporting more loaded instructions; that is, a larger on chip instruction diversity. Figure 9.2 shows how instruction density increases with increasing numbers of contexts alongside the decrease in peak computational density.

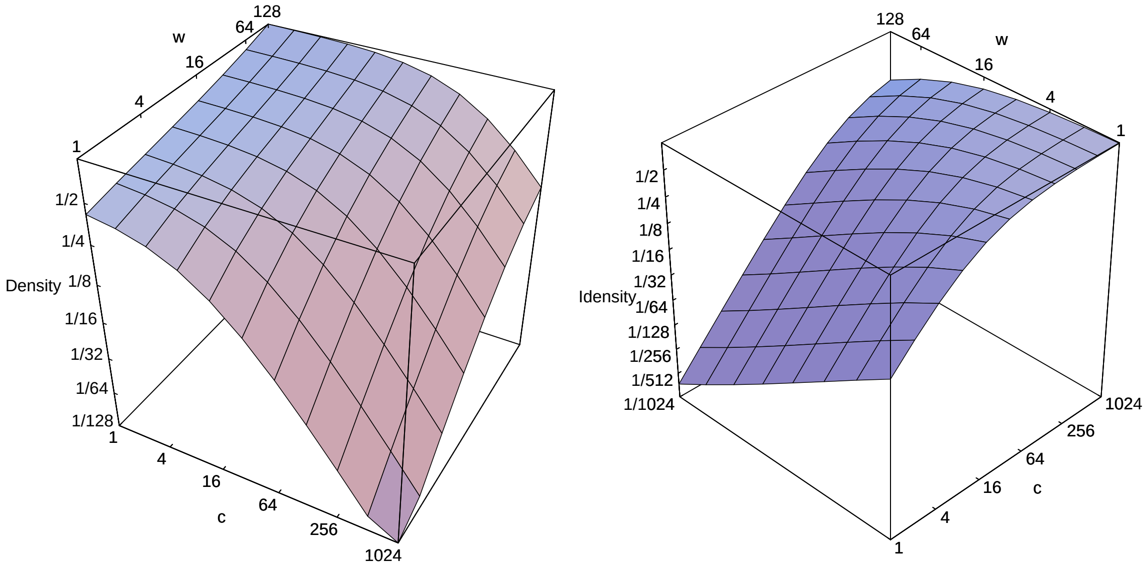
These same density trends hold if we set aside a fixed amount of data memory. The area outside of the data memory will follow the same density curves shown here.



$k = 4$, $n = 2$, $p = 0.5$, $c = d$, $N_{ctrl} = 0$, $N_p = 16384$

Reference Density of 1.0 corresponds to $w = 128$, $c = 1$

Figure 9.1: Peak Computational Density Versus Contexts and Datapath Width



Left – Computational Density; Right – Instruction Density
$k = 4$, $n = 2$, $p = 0.5$, $c = d$, $N_{ctrl} = 0$, $N_p = 16384$

Figure 9.2: Compute and Instruction Densities Versus Contexts and Datapath Width

9.3 Granularity

As noted in the previous chapter, we can use larger granularity datapaths to reduce instruction overheads. The utility of this optimization depends heavily on the granularity of the data which needs to be processed. As noted in the previous section, the coarser the granularity the higher the peak performance. However, if the architectural granularity is larger than the task data granularity, portions of the device's computational power will go to waste.

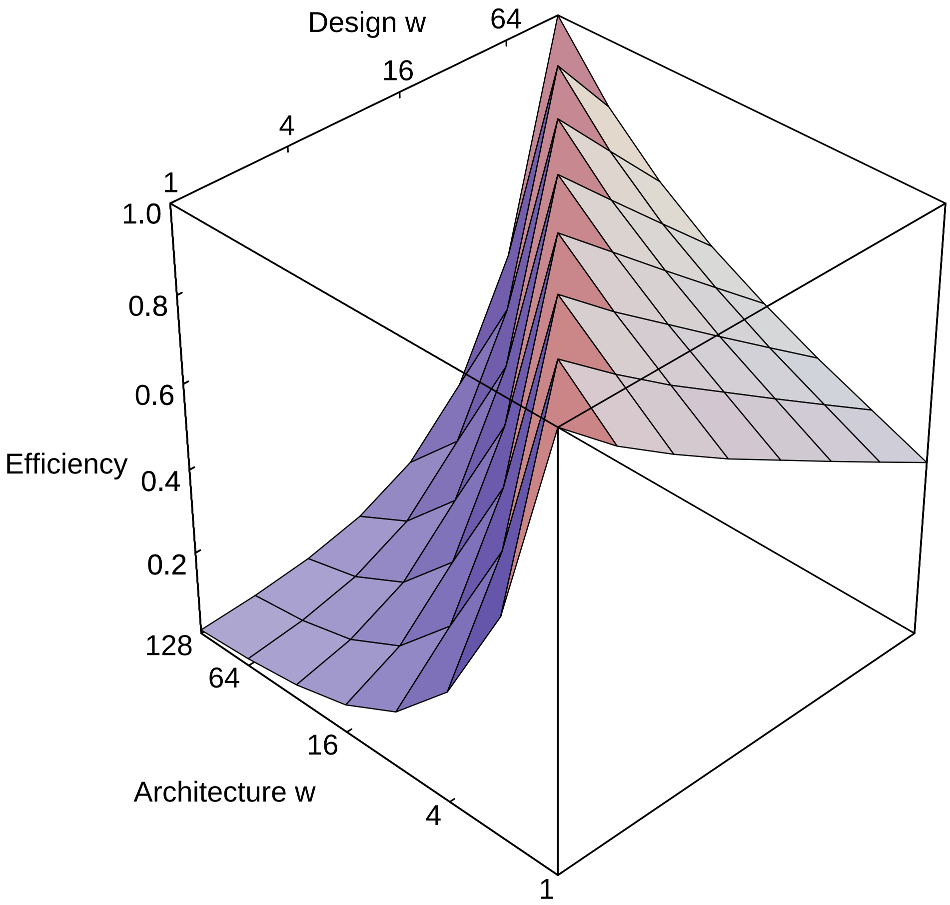
We can model the effects of pure granularity mismatches using the area model developed above. First, we note that the optimal configuration for a given word size will always be the architecture which has the same word size as the task. We can then determine the efficiency associated with running tasks with word size w_{des} on an architecture with word size w_{arch}, by dividing the area required to support the task on a w_{des} architecture by the area required on a w_{arch} architecture. For $w_{des} = C \cdot w_{arch}$, for some integer C, the efficiency is simply the ratio of the bit processing element areas. For $w_{des} > w_{arch}$, the task can run on top of the low w_{des} bit processing elements in the architecture datapath, leaving the remaining processing elements unused. The efficiency here is the ratio of the area of w_{des} bit processing elements from a w_{des} architecture versus w_{arch} bit processing elements from a w_{arch} architecture.

$$\text{Efficiency} = \begin{cases} \dfrac{A_{bit\_elm}|_{w=w_{des}}}{A_{bit\_elm}|_{w=w_{arch}}} & w_{des} = C \cdot w_{arch} \\ 1 & w_{des} = w_{arch} \\ \dfrac{w_{des} \cdot A_{bit\_elm}|_{w=w_{des}, N_p = N_{p_0}}}{w_{arch} \cdot A_{bit\_elm}|_{w=w_{arch}, N_p = \left(\frac{w_{arch}}{w_{des}}\right) N_{p_0}}} & C \cdot w_{des} = w_{arch} \end{cases} \tag{9.2}$$

Note that a single-chip implementation is assumed for comparison so that there are no boundary effects between components.

Figure 9.3 shows the efficiency for various architecture and task granularities. At $c = 1$, the active switching area dominates. The fine granularity ($w = 1$) has the most robust efficiency across task granularities. The efficiency drops off quickly for large grain architectures supporting fine grain tasks.

Figure 9.4 shows that the robustness shifts as the numbers of contexts increases. For $c = 1024$, the instruction memory space dominates the area. Consequently, the redundancy which arises when fine-grained architectures run coarse-grain tasks is quite large, leading to rapidly decreasing efficiency with increasing task grain size. In this regime, the coarse-grain architectures are more robust, since the extra datapath and networking elements are moderately inexpensive compared to the large area dedicated to instruction memory. For $c = 1024$, $w = 32$, is the most robust datapath width as shown extracted in Figure 9.5.



$k = 4$, $n = 2$, $p = 0.5$, $c = d = 1$, $N_{ctrl} = 0$, $N_p = 16384$

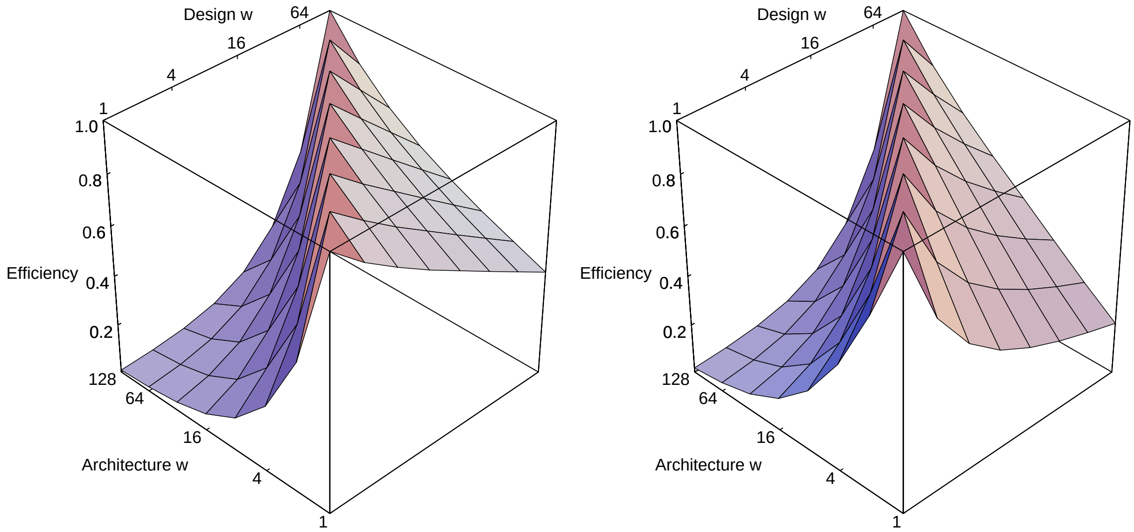
Figure 9.3: Efficiency as a Function of Architectural and Task Granularity for Single Context Architectures

These robust points correspond to the mix where the context memory makes up roughly half the area of the device.

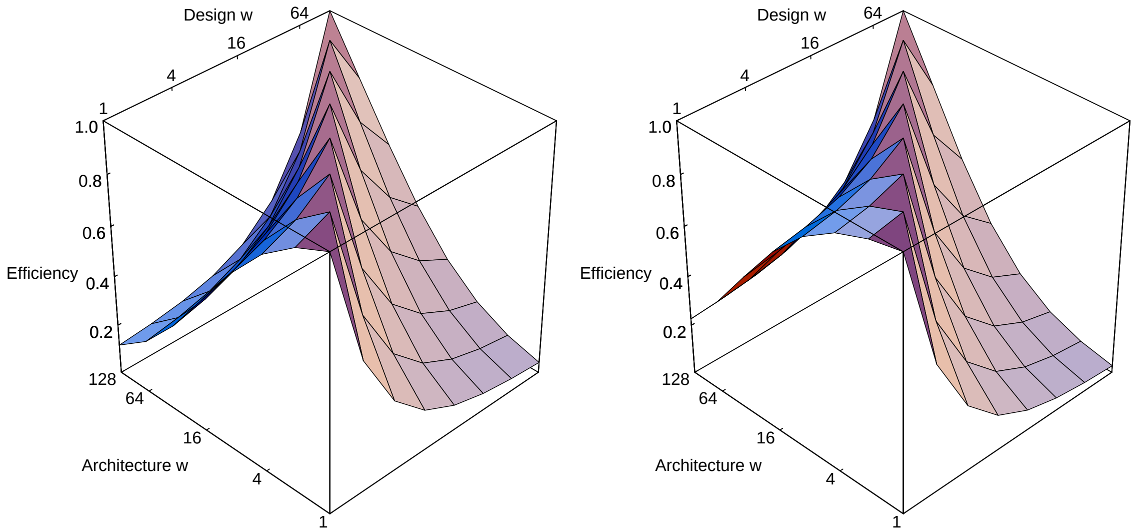
$$A_{bit\_elm}|_{c=0,w=w^*} = \left(\frac{1}{2}\right) A_{bit\_elm}|_{c=c,w=w^*} \tag{9.3}$$

At this point:

- Finer grain devices running coarser granularity tasks waste, at most, a little over half of their area – the memory area plus the switching overhead associated with finer granularity.
- Coarser grain devices running fine-grain tasks waste at most half of their area – the unused datapath area.



Left – $c = 1$; Right – $c = 16$



Left – $c = 256$; Right – $c = 1024$
$k = 4$, $n = 2$, $p = 0.5$, $c = d$, $N_{ctrl} = 0$, $N_p = 16384$

Figure 9.4: Efficiency as a Function of Architectural and Task Granularity

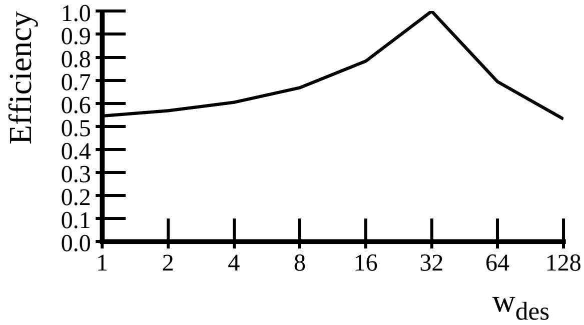


Figure 9.5: Efficiency versus Task Data Width for a 1024-context, 32-bit Granularity Device

9.4 Contexts

We saw in Section 9.2 that the computational density is heavily dependent on the number of instruction contexts supported. Architectures which support substantially more contexts than required by the application, allow a large amount of silicon area dedicated to instruction memory to go unused. Architectures which support too few contexts will leave active computing and switching resources idle waiting for the time when they are needed.

We can model the effects of varying application requirements and architectural support in an ideal setting using the area model. We assume we have a repetitive task requiring N_{op} operations which has a path length l_{path}. In an ideal packing, an architecture with $N_p = \frac{N_{op}}{l_{path}}$ processing units and $c = l_{path}$ instruction contexts can support the task optimally. If $c > l_{path}$, the area per processing element is larger than necessary to support the application. If $c < l_{path}$, it will be necessary to use more processing elements simply to hold the total set of instructions.

$$\text{Efficiency} = \begin{cases} \frac{A_{bit\_elm}|_{c=l_{path}}}{A_{bit\_elm}|_{c=c_{arch}}} & l_{path} < c_{arch} \\ 1 & l_{path} = c_{arch} \\ \frac{c_{arch} \cdot A_{bit\_elm}|_{c=l_{path}, N_p = N_{p_0}}}{l_{path} \cdot A_{bit\_elm}|_{c=c_{arch}, N_p = \left(\frac{l_{path}}{c_{arch}}\right) N_{p_0}}} & l_{path} > c_{arch} \end{cases} \tag{9.4}$$

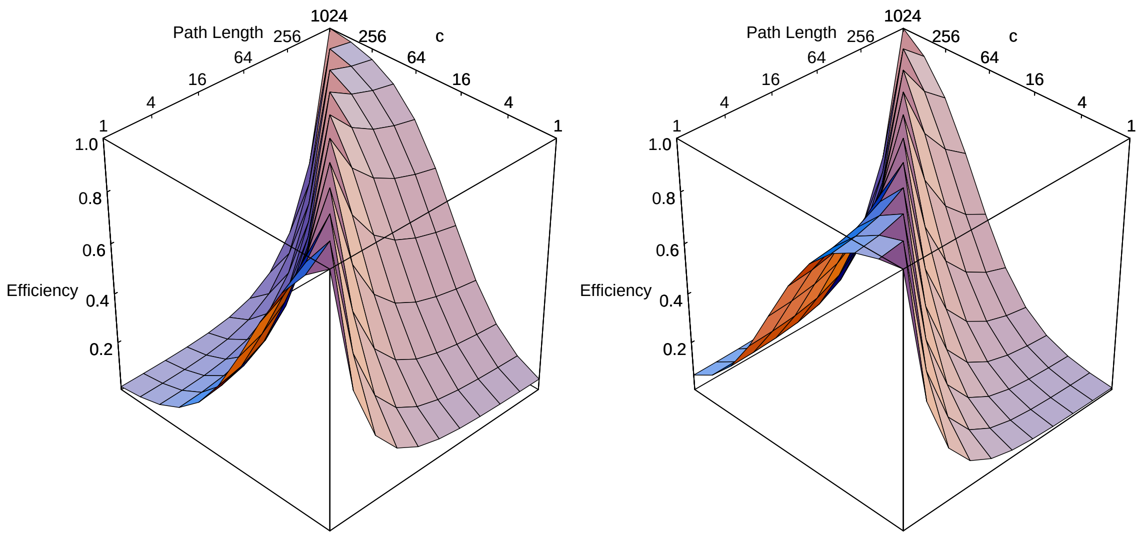
This relation is shown for several datapath widths, w, in Figure 9.6. Again, single chip implementations are assumed for comparison.

The efficiency dropoff for $l_{path} < c$ is less severe for large datapaths, large w, than for small datapaths. Similarly, the dropoff for $c < l_{path}$ is less severe for small datapaths than for large datapaths. This effect is due to the relative area contributed by instructions. In the small w case, the instruction area takes up relatively more area than in the large w case, so costs of extra active area is relatively smaller than in the large w case. In the large datapath case, the instructions make up a lower percentage of the area so the overhead for extra instructions is relatively smaller.

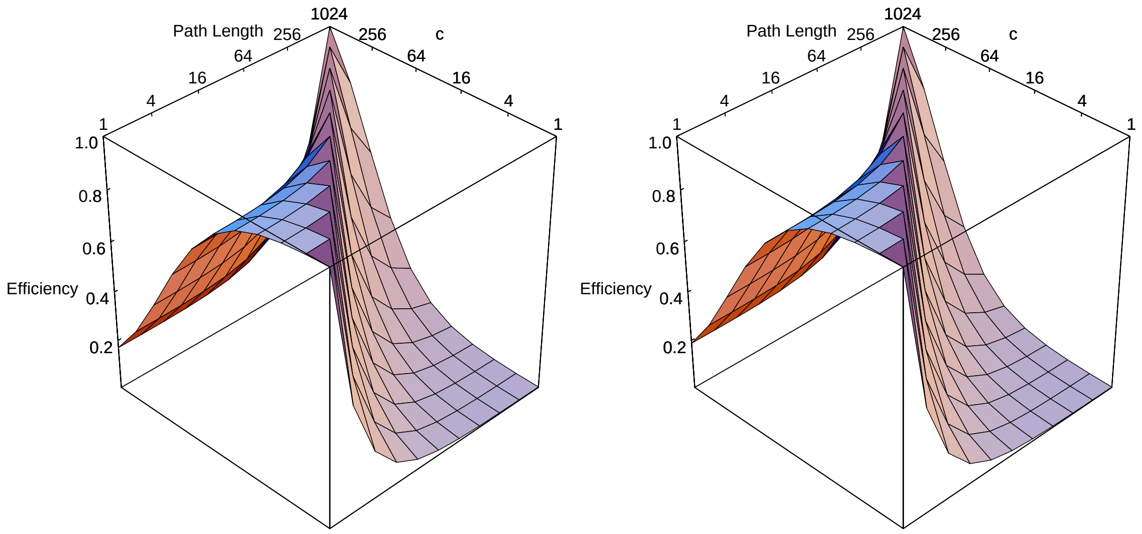
The 16 instruction context case is the most robust across this range for single bit datapaths (See Figure 9.7). Similarly, 256 instruction contexts is the most robust for $w = 128$ (See Figure 9.8). Neither of these cases drops much below 50% efficiency at either the $l_{path} < c$ or $c < l_{path}$ extremes. These "robust" cases correspond to the points where the instruction memory area is roughly equal to the active network and computing area. In either extreme, at most half of the resources are being underutilized. c^*, our robust context selection, can be defined as:

$$A_{bit\_elm}|_{c=0} \approx \left(\frac{1}{2}\right) A_{bit\_elm}|_{c=c^*} \tag{9.5}$$

Remember that the network resource requirements grow with array size. In the $c < l_{path}$ case, where we must deploy more processing elements to handle the task, the total number of processing elements increases causing the switching area per processing element to increase as well. This effects acounts for the fact that the efficiency can drop below 50% and the approximate relation in Equation 9.5.



Left – $w = 1$; Right – $w = 8$



Left – $w = 64$; Right – $w = 128$

$k = 4$, $n = 2$, $p = 0.5$, $c = d$, $N_{ctrl} = 0$, $N_p = 16384$

Figure 9.6: Efficiency as a Function of Task Path Length and Architectural Contexts

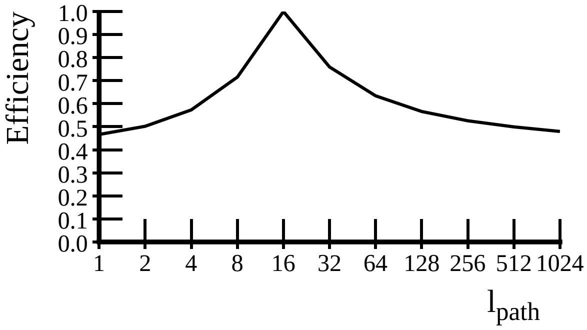


Figure 9.7: Efficiency versus Task Path Length for a 16-context, Single-bit Granularity Device

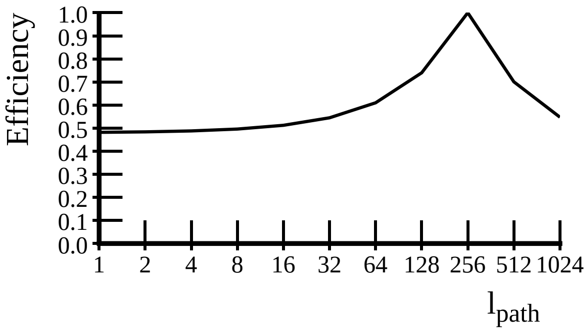


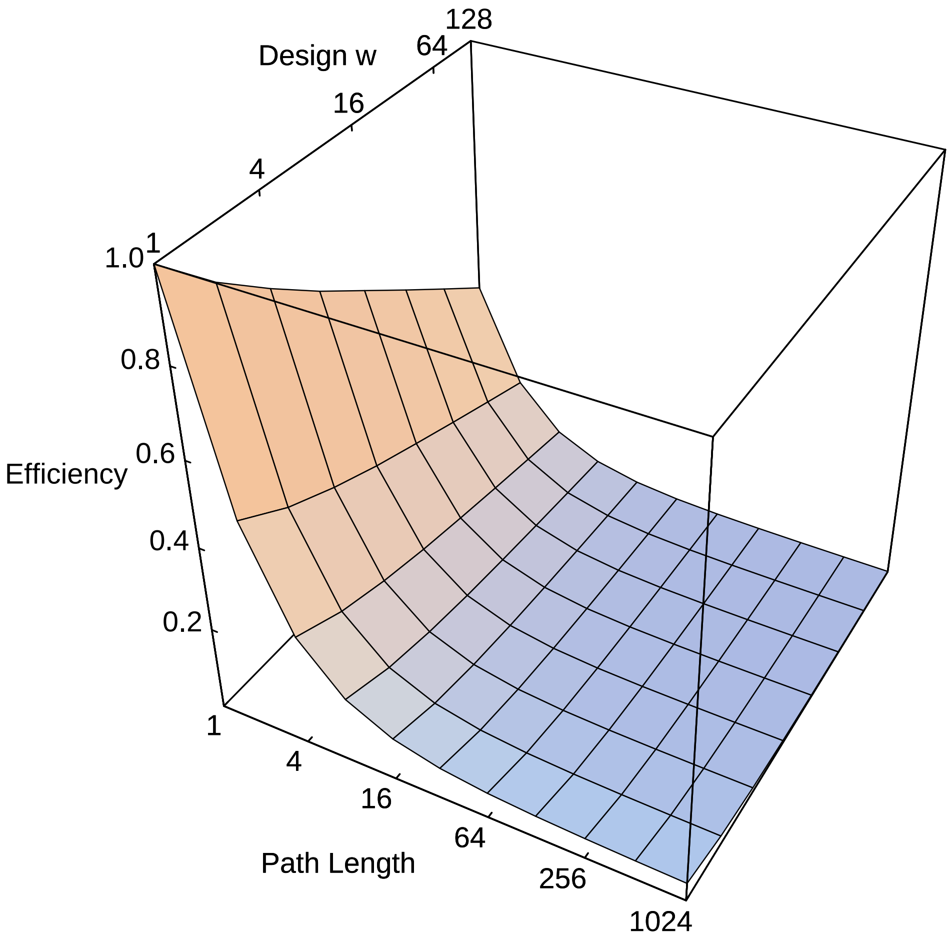
Figure 9.8: Efficiency versus Task Path Length for a 256-context, 128-bit Granularity Device

9.5 Composition

In general, we see cumulative effects of the grain size and context depth mismatches between architecture and task requirements. Figure 9.9 shows the yielded efficiency versus both application path length and grain size for the conventional FPGA design point of a single context and a single bit datapath. The FPGA drops to 1% efficiency for large datapaths with long path lengths. Similarly, Figure 9.10 shows the efficiency of a wide word ($w = 64$), deep memory ($c = 1024$) design point. While this does well for large path lengths and wide data, its efficiency at a path length and data size of one is 0.5%. *Notice here, that the wide, coarse-grain design point is over 100$\times$ less efficient than the FPGA when running tasks whose requirements match the FPGA, and the FPGA is 100$\times$ less efficient than said point when running tasks with coarse-grain data and deep path lengths.*

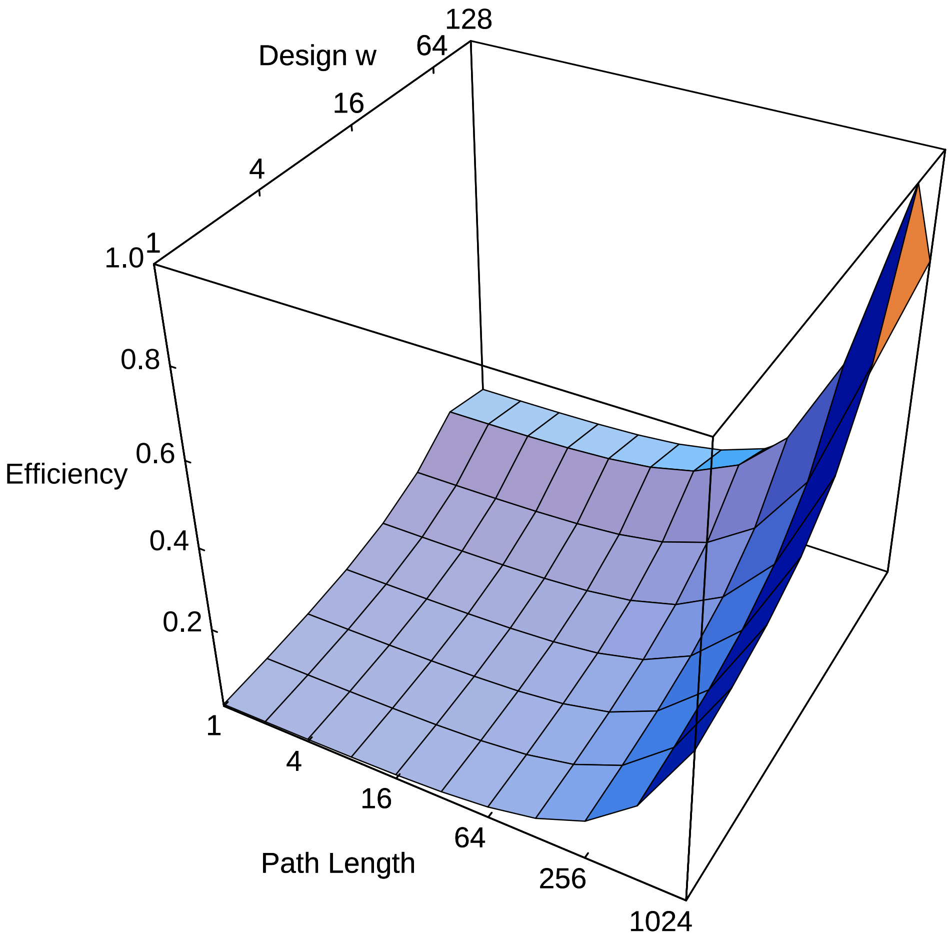
In the previous sections we saw that it was possible to select reasonably robust choices for datapath width or number of instruction contexts given that the other parameter was fixed. We also saw that the robustness criterion followed the same form; that is, the inefficiency overhead can be bounded near 50% if half of the area is dedicated to instruction memory and half to active computing resources. This does not, however, yield a single point optimum since the partitioning of the instructions between more contexts and finer-grain control is handled distinctly in the two cases.

Figure 9.11, for instance, shows the yield for a single design point, $w = 8$, $c = 64$, across varying task path lengths and datapath requirements. While the $w = 8$ and $c = 64$ cross-sections are moderately robust, the efficiencies at the extremas are low. At $l_{path} = 1$, $w = 1$, the efficiency is just under 8%, and at the $l_{path} = 1024$, $w = 128$, the efficiency is just over 8%. This design point is, nonetheless, more robust across the whole space than either of the architectures shown in Figures 9.9 and 9.10.



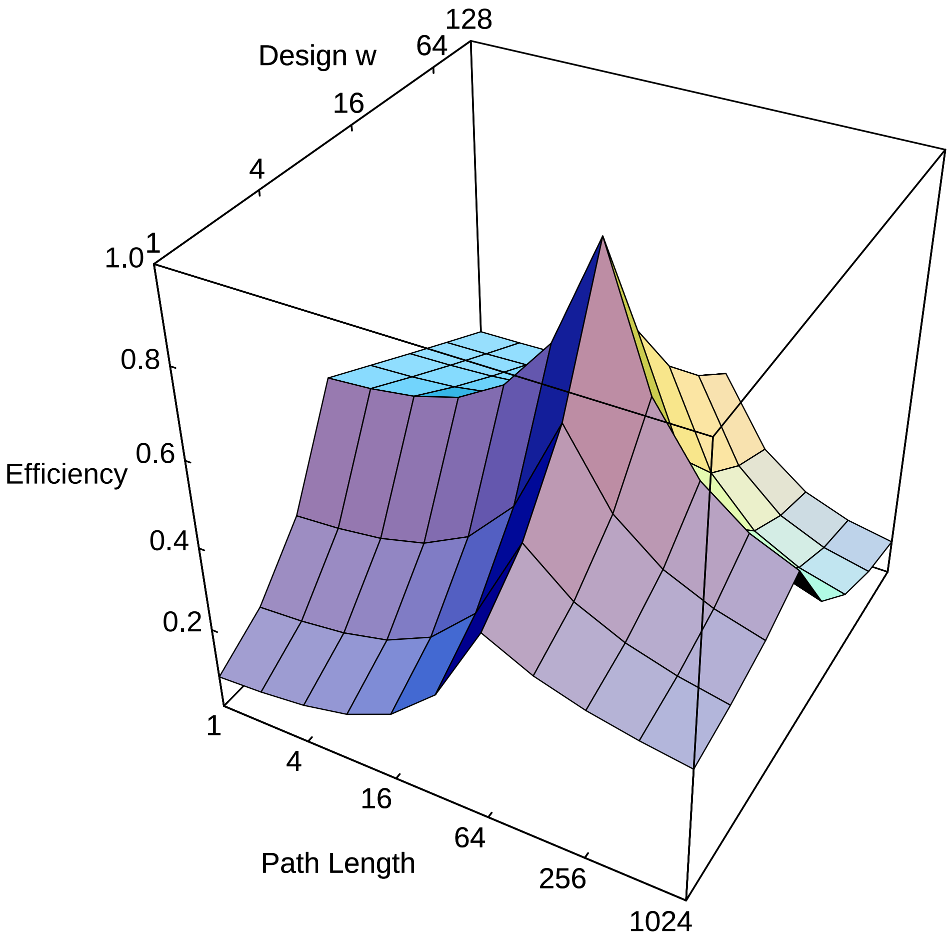
$k = 4$, $n = 2$, $p = 0.5$, $c = d = 1$, $w = 1$, $N_{ctrl} = 0$, $N_p = 16384$

Figure 9.9: Efficiency for Conventional FPGA Design Point ($w = 1$, $c = 1$)



$k = 4$, $n = 2$, $p = 0.5$, $c = d = 1024$, $w = 64$, $N_{ctrl} = 0$, $N_p = 16384$

Figure 9.10: Efficiency for Coarse-Grain, Deep Memory Design Point ($w = 64$, $c = 1024$)



$k = 4$, $n = 2$, $p = 0.5$, $c = d = 64$, $w = 8$, $N_{ctrl} = 0$, $N_p = 16384$

Figure 9.11: Efficiency for Fixed $w = 8$, $c = 64$

9.6 Summary

The area model shows us how peak capacity depends on granularity organization and instruction support. We see that the penalty for fine-granularity is moderate, 2.5× difference between $w = 1$ and $w = 128$, in the configurable domain where there is only instruction memory for a single context. The penalty is large, 36×, in the heavy multicontext domain. We also looked at the effects of application granularity and path length. In both cases, we found that, given *a priori* knowledge of either the task granularity or context requirements, we could set the other parameter such that the efficiency did not drop significantly below 50% for any choice of the unknown parameter. This is significant since the peak performance densities across the range explored differed by roughly a factor of 200×. For both of these cases, the robust selection criterion is to choose the free parameter such that instruction memory accounts for one half of the processing cell area. We saw that the effects of granularity and path length mismatches were cumulative and that FPGAs running tasks suited for deep memory, coarse-grained architectures can be only 1% efficient. If we must select both the datapath granularity and the number of contexts obliviously, we cannot obtain a single design point with as robust a behavior as when we only had one free parameter. A good design point across this region of the *RP*-space suffers a 13× worst-case overhead.