A Design Time Simulator for Computer Architects

Sangeetha Sudhakrishnan, Francisco-Javier Mesa-Martinez, and Jose Renau
Dept. of Computer Engineering, University of California Santa Cruz
Email: sangeetha, javi, renau@soe.ucsc.edu

***Abstract*—Processor design and implementation is a complex and resource intensive enterprise. Ideally, designers should be able to quantify the design time implications of their architectural proposals, in order to make more educated decisions and design trade offs. To address the lack of quantitative methodologies to estimate processor design time, this paper introduces a new class of event-driven simulation: μDSim.**

Our proposed simulation infrastructure models the interaction between engineers during the development and verification cycles. To validate μDSim, we compare estimated design times against data gathered during the development and verification of three different academic processors and three industrial multiprocessor systems. As an example application for the architectural community, we estimate the design time for a previously published architectural proposal.

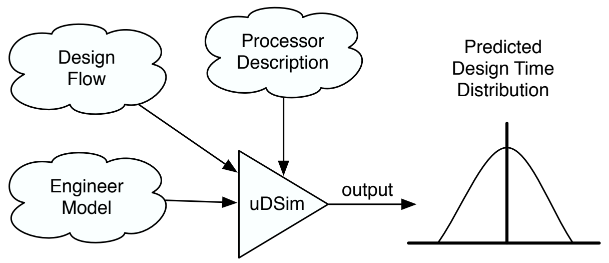

Fig. 1: μDSim infrastructure.

figure

I. Introduction

The design of a modern processor is a extremely resource intensive endeavor. Yet there has been little work [1] done to measure, understand, and estimate the effort associated with processor design. Different competing designs may not only vary in their power and performance characteristics, but they may also differ in their design times. Since alternative changes in a given architectural element may impact the project differently, designers as well as managers should be able to estimate their design times. Making architectural decisions without quantitatively considering their impact on the design time can lead to designs which are too complex. Such lack of insight may lead to missed processor design deadlines, a major cause for the increase in costs, and sometimes even to the cancellation of design projects, like the Sun Millenium [2].

This paper proposes a simulation infrastructure (μDSim) that predicts the design time required for a given processor description. Such simulation infrastructure needs to solve three main problems: How to create a generic model of the engineers (Engineer Model); How to specify a processor before it is implemented (Processor Description); How to model the design flow commonly found in processor design teams (Design Flow).

To obtain the expected design time, μDSim models several pseudo-random possible design time lines. Similar to architectural simulators, it does so by advancing through all the development phases in small quanta of time. The Engineer Model generates pseudo-random events (communication) as the time advances based on the processor specification. μDSim performs several simulations to capture the multiple random factors at play when modeling design processes. This has the advantage of providing a distribution of design times, with the more unpredictable designs having wider distributions.

μDSim leverages metrics and studies from software engineering, with significant differences that are specific to processor designs. μDSim uses an simulation model instead of an analytical model. Several studies [1, 3, 4] propose relatively simple equations to estimate design effort. There are two key problems with such systems that μDSim addresses: (1) per team parameters; (2) methods to estimate lines of code (LoC). The per team parameters imply that to estimate the design time for a given project, a per team parameter needs to be found [1]. This complicates the whole process because some parameters are not known until the project is finished, thus methods to estimate these parameters in advance are required. Requiring the lines of code (LoC) is equally challenging, software engineering models [3, 4] dedicate most of their efforts on how to estimate them. The use of cyclomatic complexity, for our processor model, greatly simplifies these issues due to its composability and because it is easily estimated by architects.

The resulting μDSim simulation infrastructure makes it possible, for the research community, to compare the relative complexities of different processor components. μDSim enables designers to quantitatively evaluate the design times for their proposals. As the evaluation shows, μDSim achieves a 0.98 correlation predicting design times, which is a clear improvement over previous analytical work [1].

μDSim can be used to estimate not only the overall time to complete an entire processor, but also the time required to implement alternative architectural proposals. As an example of usage, the evaluation section shows that SEED [5], a processor issue logic mechanism with significant improvements in performance, power, and area, displays also a significant impact on design time. μDSim reports that an Alpha-like processor with a SEED scheduler has a 10% design time increase when compared against the same Alpha-like processor with a traditional scheduler. In a way, this 10% design time increase is equivalent to a net slowdown of 8% due to being later to market if we assume that processors double performance every 2 years.

II. μDSim Simulator

μDSim is a simulator that evaluates the impact, in design time, of specific architectural proposals. Its modus operandi is similar to most event-driven architectural simulators. Instead of dealing with events such as memory access or processor clock cycle advances, μDSim deals with events that model communication between engineers, their productivity, and estimates the design time for a given engineering organization based on a high level description of the design to be evaluated.

The simulation infrastructure allows to easily modify the design specification for the processor being evaluated. The number of engineers and design teams are fully configurable. Thus allowing architects to simulate the design and implementation characteristics of their proposals under different design team configurations. Figure 1 shows the μDSim simulator with its three major components: engineer model (Section A), processor description (Section B), and design flow (Section C). Each of the major components of the simulator are defined in the sections that follow.

Predicting the exact finishing time/date for a project is a challenging task. This is due to the fact that design processes involve many factors which are very difficult to estimate accurately, such as productivity differentials between engineers and their communication. To account for this uncertainty, μDSim performs many simulations for a given project, with several engineer parameters using Gaussian distributions with defined mean and standard deviations. By performing multiple simulations, the output of μDSim is not a discrete value, but a distribution for the estimated design time. Intuitively, projects with a smaller standard deviation are more predictable.

II.A. Engineer Model

Humans, and therefore engineers, seem difficult to model because of unpredictable random events (illness, world events, lost of data...) and the large amount of parameters that can affect their productivity. After all, there are many books just to address parameters like cubicle size, team organization, and tool productivity. These issues seem to imply that μDSim needs to consider a very large set of parameters to accurately predict the design time. To deal with the possible complexity, our approach uses the minimum set of parameters possible to achieve a good correlation with the observed data for the real processor projects in our study, while still capturing well known software engineering principles properly.

TABLE I: μDSim engineer parameters. Sync, Async, and Comm stand for synchronous, asynchronous, and communication respectively.

table

Name		Source	Value
Skill Level	Mean	Fitted	$0.2\ \frac{\text{Cyclomatic}}{\text{hour}}$
	Std. Dev.	[6]	5 times
Skill Learning		[7]	x2.2 in 15 months
Team Size.	Mean	[6]	7 engineers
	Std. Dev.	[6]	3 engineers
Sync. Comm.	Mean	[8]	$104\ \frac{\text{minutes}}{\text{day}}$
	Std. Dev.	[8]	95 minutes
Async. Comm.	Mean	[8]	$35\ \frac{\text{minutes}}{\text{day}}$
	Std. Dev.	[8]	53 minutes

Table I shows all the engineer parameters used by μDSim. This table only has five different parameters, some of them follow a Gaussian distribution (Mean, Std. Dev.). Four of these parameters are obtained from previous publications. Only one parameter (mean Skill Level) is tuned using a subset of the processors available as a training set. As the evaluation shows, we have 6 processors available with multiple breakdowns in design time. In total, we have 20 different design components with their respective design times and team sizes. We use 3 processors (10 components) to tune just the mean skill level.

Skill Level and Learning

Initial studies in software engineering conclude that not all engineers are equal [9]. The differential in skill levels across engineers in a single project can be as high as 1 order of magnitude. In our simulation infrastructure, the skill levels of engineers have a Gaussian distribution (mean Skill μ, std. dev. Skill σ). The average skill is obtained by performing a regression over the training set. Furthermore, the standard deviation is based on the observation that it is common to find up to 5 times difference in productivity [6].

Technical skills do not stay constant with time either [7], the longer engineers work in a specific project (Proficiency Time) the higher their productivity. Previous work observed productivity increases of up to 120% in 15 months [7]. μDSim introduces a skill learning curve which applies the following correction:

SSL = Starting Skill Level

$$\text{Skill} = \begin{cases} 2.2 \times \text{SSL} & \text{time} > 15 \text{ months} \\ (1{+}1.2 \times \frac{\text{Time}}{15\text{ months}}) \times \text{SSL} & \text{otherwise} \end{cases}$$

Team Size

The design team in this study is represented as a balanced tree of manager and engineers. Each parent node represents a manager, which has a set of leaves representing the engineers or sub-managers under its command. Each manager is assumed to have a Gaussian distribution with a mean of 7 engineers under its command, and a standard deviation of 3 engineers. These parameters have been approximated from previous studies [6].

Communication

Communication and interaction between engineers are fundamental aspects in any design environment. Overall efficiency decreases as the number of engineers in a single project increases [6, 9, 10]. This is due mostly to the communication overhead associated with the size of a design team. When an engineer works in a project he/she needs to know the project description, learn the development environment, familiarize with the design methodology, and many other factors. Communication does not only decrease the engineer's time available to work on the project but also affects (reduces) the efficiency of the person at the other end of the conversation.

There are two kinds of communication considered in our model: Synchronous and Asynchronous. Synchronous communications are interactions between the team members. This includes meetings and other events where there is a face to face communication. Asynchronous communications mostly corresponds to emails, documentation, collaboration through repositories, etc.

μDSim randomly generates synchronous and asynchronous events as each engineer advances his/her design. While asynchronous communication does not interrupt other workers while busy, synchronous communication does. μDSim uses the communication parameters and values shown in [8]. Different companies may have different communication patterns, in this work however, we assume that all the projects use the same communication parameters.

II.B. Processor Description

In order to estimate design time, we must define what elements are needed to provide a complete description for a processor. Processors have many components or units that interact with each other. For example, the load/store queue interacts directly with the data cache, but not with the instruction cache. The components existing in a processor (or multiprocessor) can be seen as a graph where each node provides some basic functionality, and the edges are the interactions between components. While the edges represent interactions between units, the nodes represent the units. Each node is labeled with a complexity metric. The higher the value the harder it is to design the unit. Nodes are undivisible, this means that a single person should work on it. Typically, a manager or architect extracts a set of connected nodes and assigns them to an engineer.

The rest of this section describes how to estimate the overall complexity for each node (Component Complexity Estimation), how processor designs can be partitioned into several nodes (Component Selection), and the partitioning algorithm used by the simulated managers to assign tasks to engineers (Design Partition).

Component Complexity Estimation

[1] demonstrates that Lines of Code (LoC) is better estimating processor complexity than synthesis metrics. However, LoC are difficult to estimate. The Cyclomatic complexity, also known as the McCabe number [11], is an alternative software metric to estimate design complexity. Assuming a program can be represented as a control flow graph, the Cyclomatic complexity for a program is defined as the number of edges minus the number of nodes plus one. More intuitively, the Cyclomatic complexity is equivalent to the number of linearly independent paths which exist in a given program. Which are, incidentally, exactly the same as the minimum number of tests required to provide a full testbench coverage.

Intuitively, Cyclomatic complexity captures the connectivity displayed by the control flow graph. Although Cyclomatic complexity was originally developed for software projects, it is also applicable to processor designs using collections of Hardware Description Language (HDL) statements. In theory, the number of statements in a program and the Cyclomatic complexity can be very different, but previous work [12] has shown that Cyclomatic complexity and the lines of code are highly correlated for HDL programs. Since lines of code and number of statements are two of the best metrics to estimate design effort [1] on microprocessors, Cyclomatic complexity is a good alternative metric. The evaluation section in this paper performs simulations to verify the previous assumption.

Cyclomatic complexity also has the advantage that it is easy to estimate by computer architects. Intuitively, if we have a stateless component with four different types of responses for the given inputs, its Cyclomatic complexity is four. For example, the Cyclomatic complexity for a simple Arithmetic Logic Unit (ALU) is equal to the number of operations implemented. For a more complex block like a Power4-like [13] Store Reorder Queue (SRQ), we enumerate the linearly independent behaviors that the SRQ has. If we ignore the multiprocessor reply/flush triggers, the Load Reorder Queue (LRQ), and Store Data Queue (SDQ) updates, the SRQ has the following situations or behaviors to be implemented and verified: reset management, stores allocate an SRQ entry reservation at rename, store address update at execution, store completion at retirement, find the correct SRQ for store-load forwarding, perform word, half-word, byte load or no forwarding at load execution, flush triggered when only partial load forwarding is available, load replay triggered when the address is in the SRQ but still not in the SDQ, and SRQ full signal management. Since there are 12 linearly independent cases, we have a 12 Cyclomatic complexity for the Power4-like SRQ unit.

Component Selection

Given any processor structure, we can create further recursive decompositions until the basic cell operations are reached. Another key advantage of Cyclomatic complexity is that it is a composable metric. This means that if we have a unit with a given Cyclomatic complexity (A), we can divide it in two units (B and C) where the $\texttt{Cyclo}(A) = \texttt{Cyclo}(B) + \texttt{Cyclo}(C)$.

Once a manager is able to create enough tasks to keep all the engineers busy, there is no reason to subdivide components any further [1]. For example, the evaluation shows that clustering graph nodes has little effect on the overall design time as long as the number of engineers is significantly bigger than the number of components. To avoid significant unbalances in the graphs, we try to minimize the difference in Cyclomatic complexity between graph nodes.

It is necessary to partition the processor into multiple high-level components. The number of components ($\#C$) should be an order of magnitude larger than the number of engineers ($\#E$). To estimate the Cyclomatic complexity for each component (C_i), it is necessary to estimate the minimum number of linearly independent behaviors. If the Cyclomatic complexity for any component (C_i) is an order of magnitude bigger than the average complexity ($AveC$), then we further divide that component and recalculate the complexity for the sub-components. For large projects, the $AveC$ can be fairly large. Nevertheless, the same software engineering recomendations about Cyclomatic complexity hold for a processor HDL codebase. The original recomended Cyclomatic complexity limit is 10, some other researchers suggested higher limits, however blocks with over 30 Cyclomatic complexity are considered very difficult to verify or to correctly estimate their Cyclomatic complexity. This is why we recommend to further partition a block if its C_i is over 30.

Design Partition

As stated previously, a project can be represented by a graph where each node has a defined Cyclomatic complexity. The goal of the project manager becomes assigning tasks to engineers so that all the engineers can be busy, thus maximizing their productivity. To avoid communication overheads, the manager partitions the project graph in several sub-graphs and assigns components to engineers based on their skills. Based on [1], we know that for single-engineer projects, there is a linear dependence between project size and time to complete the task. Therefore, the manager may use graph partitioning techniques to minimize communication between tasks and approximate the completion time based on the Cyclomatic complexity for that component.

The graph partition algorithm utilized is very similar to VLSI

[1] Leaving very complex single nodes may lead to incorrect Cyclomatic estimation because it is easier to miss some different behavior. The level typically used by architects is appropiate because it needs to explain the complete behavior or operation of each unit.

floorplaning algorithms. μDSim uses Metis [14] which performs simulated annealing to partition the graph in a balanced way while minimizing edges between partitions. The manager partitions the design based on the number of engineers working on the project.

II.C. Design Flow

The scope for the design flow considered by the simulation infrastructure in this paper is limited to a timeline inspired by software engineering projects, which are commonly divided in 4 phases: drafting of requirements, design description, implementation coding, and testing.

Typical software engineering projects break their timelines into 20% dedicated to requirements, 20% design, 20% coding, and 40% for testing. This breakdown is obtained after multiple studies which have shown that, contrary to common assumptions, testing tends to require double the time than coding. This is true for both logic designs [15] and software engineering [9]. μDSim requires a description of the project. Therefore, the component indicating the requirements phase is not included for consideration. As a result, the actual timeline modeled is; design, coding, and testing.

If we ignore the communication overheads included on the engineering team model, and assume that there is a single designer, the three phases of the design flow (design, coding, testing) are highly correlated with Cyclomatic complexity. Furthermore, as stated above, Cyclomatic complexity is equivalent to the number of different paths present in the program. This is exactly the same as the minimum number of testbenches required to have a 100% coverage. Thus, the information related by the Cyclomatic complexity or the minimum number of tests required is highly correlated with the time required to test a module.

Previous work shows that the number of statements [1] is a good metric to estimate design effort for several processors. However, that study does not consider Cyclomatic complexity as a possible metric. The evaluation section shows that Cyclomatic complexity is as correlated with design effort as the number of statements, by analyzing the same processor designs in [1]. As a result, we use Cyclomatic complexity as an approximation to design effort if overheads are ignored.

The last major component of the design flow is the actual design phase. Intuition suggests that Cyclomatic complexity should be correlated with the design time for HDLs, however we do not have empirical data to backup this claim. Nevertheless, since there are previous studies [9] which show a specific percentage breakdown between design and coding/testing, a correct estimation for testing can be used to also extrapolate design time. Both testing and coding are highly correlated with Cyclomatic complexity. Therefore it seems safe to assume that design time is also highly correlated with Cyclomatic complexity.

Figure 2 shows an example of a generic design flow. The manager partitions the design in several sub-tasks. Since we follow an Agile development process, the manager tries to extract tasks that require 1 month work of design/coding. As new tasks are created (A,B,C,D) any free engineer in the project group can be assigned by the manager to work on the new task. As the tasks complete, depending on the skill level of the engineer he/she may or may not select a new task.

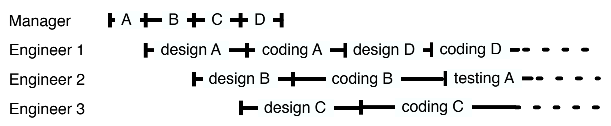

Fig. 2: Sample design flow Gantt chart for 3 engineers.
figure

The infrastructure is flexible enough that multiple design flows may be defined and simulated. For example, companies like Intel utilize two competing teams working on the same project in order to reduce variability. μDSim can be extended to model such design flows. In this paper, we just focus on developing the simplest possible infrastructure that can be successfully validated using the data obtained for several real processor projects.

III. Evaluation

The evaluation for μDSim is divided in five parts. We start by showing the simulation results for two processors (Section A). We then analyze the overall accuracy for the time estimates provided by μDSim, by comparing these estimates against the real design times for several processors (Section B). We conclude by showing an example application for our simulation framework, where the design time estimation is added to the evaluation of the design of an out-of-order issue logic (Section E).

III.A. μDSim Simulation Results

Each μDSim simulation result reports a possible design time. Randomized factors like different productiveness between engineers, different work distributions, and communication times imply that a single μDSim simulation is not enough to estimate the expected design time. To obtain a representative design time distribution, we perform 200 simulations for each project in this paper. Although μDSim is implemented in Ruby, it requires less than 20 minutes to do all the Monte-Carlo simulations for any evaluated processor.

Figure 3 shows the μDSim simulation histogram when modeling the design time for the Illinois Verilog Model [16] (IVM) with a single engineer. The x-axis shows the design time in days, it is divided in 5 quantiles (0Q, 1Q, 2Q, 3Q, and 4Q). The graph also includes two additional vertical lines for the mean and the reported design time for IVM (460 days). Similarly, Figure 4 shows the μDSim histogram for the Sun OpenSPARC processor (1st version of Sun Niagara). For the OpenSPARC, the design time reported by Sun is 440 days with approximately 32 engineers working.

Neither of the plots shows a Gaussian or normal distribution. While the IVM looks like a log-normal distribution with a positive skew, the OpenSPARC has a bimodal distribution. Different problem partitioning and communication overheads create different distributions.

Since each simulation can have a different distribution, we can not use standard deviation or confidence intervals. Instead, we report the median design time [2], and the 1st and 3rd quantile. For IVM, the median is 536 days, the 1st quantile starts at 473 days and the 3rd quantile finishes at 620 days. We summarize this information as a ternary (473, 536, 620) which we refer

[2] The median corresponds to the 2nd quantile.

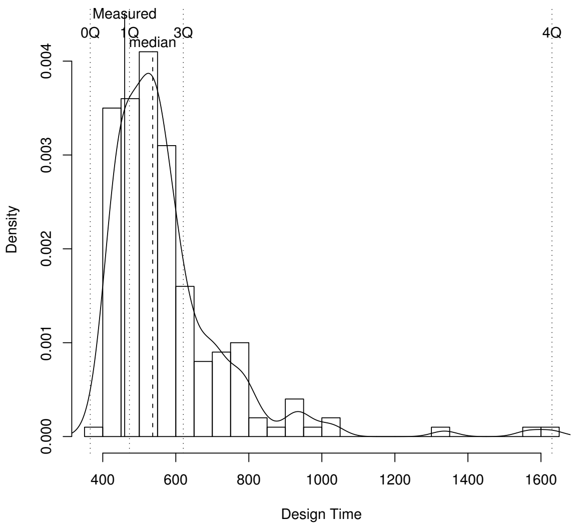

Fig. 3: IVM design time histograms reported by μDSim.
figure

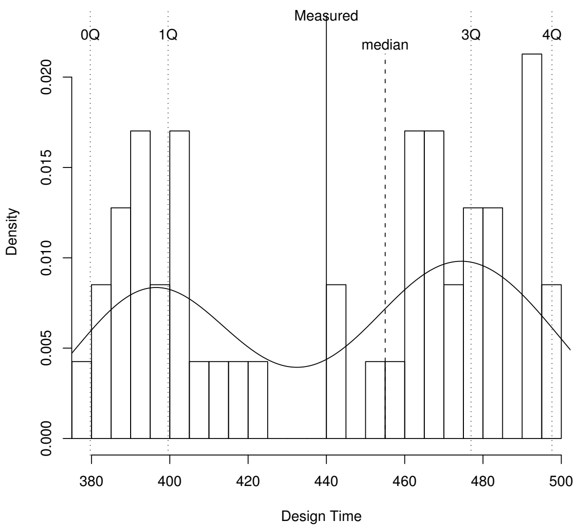

Fig. 4: Sun OpenSPARC design time histograms reported by μDSim.
figure

to as a μDSim prediction. For IVM, it means that 50% of the μDSim simulations finished between 473 and 620 days, with a median of 536 days. In contrast, for OpenSPARC we have a (400, 454, 478) μDSim prediction.

III.B. Overall Time Estimation

To evaluate the accuracy of the estimates provided by μDSim, we gather the design times for multiple real-world commercial and academic processor designs. In all the cases we asked the designers for the design times including design and verification. Whenever possible, the designers provided a breakdown by components. The design data gathered from academic processors includes two out-of-order processors (PUMA, IVM), and one vector-thread processor (SCALE). We also report design times for three Sun Microsystems processors: OpenSPARC, Niagara and Victoria Falls.

PUMA is a taped-out two-issue superscalar processor which implements a subset of the PowerPC integer instruction set (ISA). PUMA is part of a study by the University of Michigan on high-performance processors implemented in radiation-hardened GaAs processes. The Illinois Verilog Model (IVM) implements a subset of the Alpha 21264 architecture. The cache hierarchy is not modeled. IVM is part of a research project in fault-tolerance at the University of Illinois. Both projects are fully synthesizable using the Verilog HDL. SCALE [17] is a low-power, high performance embedded processor, designed at the Massachusetts Institute of Technology. The processor follows a vector-thread architectural paradigm and has been taped-out.

The OpenSPARC processor is based on the published Verilog implementation for the Niagara 1 (T1) released by Sun. Niagara 2 (T2) is a multithreaded, multicore CPU. Victoria Falls (T2 Plus) is based mostly on the Niagara 2 processor, and extends it to provide multi-chip coherence. Sun has provided us the design times and the number of engineers in the front-end design and verification teams. OpenSPARC required 2 years approximately with 32 engineers, the Niagara 2 required over 3 years with 40 engineers. Finally, the Victoria Falls project required a year and the same number of engineers.

The inclusion of such varied processor designs in our study, helps in validating the utility of μDSim for both industry and academia. All the designs considered are implemented in Verilog. The number of statements and the Cyclomatic complexity for each Verilog file is extracted automatically via a custom parsing tool. The tool also builds the component graph extracted for each Verilog file.

Table II shows the main results. Each row represents either a processor block, or the whole processor design itself. Each component in the table for which the design times are known is marked with a *. The values for design time and number of engineers are obtained from the designers. Design time reflects the total "wall clock" time it took to complete each design. All μDSim simulations use exactly the same design flow and engineer model parameters, only the number of engineers and the processor description vary across simulations for the different processors.

The 2nd column of Table II shows the number of engineers used while computing the mean design time of a component. The 3rd column (Measured Design Time) is the measured time in days [3] that it takes to implement each discrete processor component, as obtained from [1]. The 4th column shows the predicted median times to complete the design as explained above, the 5th column shows the minimum time for each design, the 6th and 7th columns show the first and third quantiles respectively, finally the 8th column shows the maximum design time.

In order to validate μDSim, we report the correlation factor (τ) between the estimated and the reported design times for each component. Correlation provides a standard method to quantify the relationship between two sets of values. Cohen [18]

[3] The evaluation assumes a 8 hour work day, 40 hour week, 160 hour month, and 2000 hour year which assumes no holidays.

TABLE II: Design time estimation for different processors and team sizes. # Eng. stands for number of engineers.

table

Module	# Eng.	Measured Design Time	Predicted Median Time	Min	First Quantile	Third Quantile	Max
[4]PUMA-Fetch*	1	60	44	28	39	52	93
[4]PUMA-Decode*	2	80	113	25	102	127	183
[4]PUMA-ROB*	1	80	130	81	114	156	237
[4]PUMA-Core*	4	240	206	93	188	227	321
[4]PUMA-Memory*	1	20	29	19	26	34	62
PUMA	1	N/A	558	380	500	631	1202
PUMA	2	N/A	393	192	360	441	621
[4]PUMA*	4	240	236	126	174	258	375
IVM-Fetch*	1	121	110	75	98	131	259
IVM-Decode*	1	24	31	19	27	35	111
IVM-Rename*	1	48	75	51	65	86	214
IVM-Issue*	1	48	77	51	69	91	196
IVM-Execute*	1	36	76	48	65	89	238
IVM-Memory*	1	122	237	152	211	273	791
IVM-Retire*	1	61	85	56	73	102	217
IVM *	1	460	536	365	473	620	1630
IVM	2	N/A	384	201	356	423	549
IVM	4	N/A	249	111	226	270	417
OpenSPARC	4	N/A	791	383	733	851	1136
[4]OpenSPARC*	32	440	454	375	400	478	536
[4]SCALE-Cache*	1	180	195	121	174	222	529
[4]SCALE-Core*	1	380	464	310	412	557	1222
SCALE	1	N/A	546	410	704	611	1502
[4]SCALE *	2	380	424	148	392	476	805
SCALE	4	N/A	259	107	237	284	637
Victoria Falls	26	N/A	396	379	392	401	484
Victoria Falls*	40	360	399	377	393	411	456
Niagara 2*	40	720	693	662	682	703	978
Niagara 2	99	N/A	678	648	672	686	734
Correlation (τ)	N/A	1	0.981	0.21	0.982	0.96	0.69

classifies correlation values in the following ranges: 1 perfect, $1 > r > 0.9$ nearly perfect, and $0.9 > r > 0.7$ very large. In Table II all the components that were used to calculate the correlation are marked with a *.

We use a subset of the processors (PUMA, SCALE, and OpenSPARC) to tune the skill level μDSim parameter, with 10 components overall. The subset of processors used in the training set in marked with [4] in table II. These processors are in the training set because they are very diverse; while PUMA and SCALE are small academic processors, SCALE is 1.2 times more complex in terms of Cyclomatic complexity than PUMA and requires 7 months longer to fully implement with 2 extra engineers.

On the other hand OpenSPARC, an industrial processor, has four times greater Cyclomatic complexity than PUMA and three times greater Cyclomatic complexity than SCALE. The correlation when only the training set is used is 0.98. When μDSim applies the skill level learned from the training set (PUMA, SCALE, OpenSPARC) to the testing set (IVM, Niagara 2, Victoria Falls) the correlation remains at 0.98. This indicates that the quality of the estimate is very good.

For this study, μDSim obtains an average correlation value of 0.98, which is very large. For comparison purposes, the design time predicted by [1] for their DEE1 metric is 0.86. DEE1 assumes complete knowledge of the design and requires productivity adjustments for each project. In contrast, μDSim only requires a single parameter to be tuned (skill), since all the other parameters are obtained directly from previously published studies. As a result, we feel that μDSim is a significant improvement towards developing reliable quantitative approaches for the estimation of design time.

[4] The subset of processors used in the training set is marked with a [4] in table II.

III.C. Model Justification

The previous section compares μDSim with specific processor design times, we now proceed to target commonly accepted software engineering models for comparison. Modern processors designs rely on HDL languages like Verilog and VHDL. Therefore, trends found in software engineering (SE) projects should hold true with μDSim. This section focuses on the analysis of our simulation infrastructure with respect to three SE trends: Project size impact, design team impact, and the Brook's Law [6]. μDSim does not directly encode any of these three trends. To our knowledge, this is the first time that they are validated against a simulation infrastructure.

Project Size Impact

Project size has a super-linear relationship with design time. Even for projects involving a single engineer, doubling the project size more than doubles the design time. Thus, as the size of a project increases, productivity decreases.

Analytical software models like COCOMO [3] and FPA [4] capture this observation. COCOMO approximates design effort with $a * \text{Size}^b$, where a is a project based constant and b is an exponent bigger than one. The more challenging the project the bigger the exponent. COCOMO suggest the use of a 1.05 exponent for simple "organic" projects, and 1.2 for "embedded" projects. We can not find approximate values for HDL projects, so we assume that HDL projects are as complex as the most complex embedded projects.

Figure 5 plots the design time for different project sizes. The data in this figure reflects a single engineer working on multiple projects of varying size. To capture the design time, we

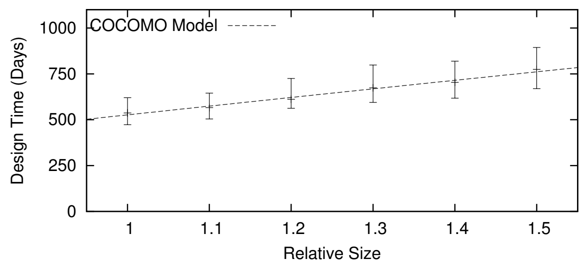

Fig. 5: Design time for different project sizes assuming a single engineer.

figure

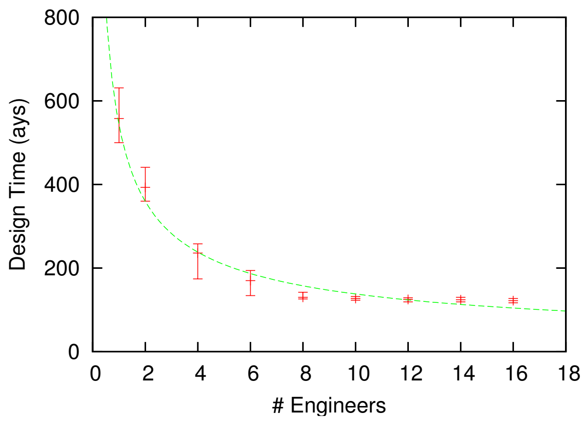

Fig. 6: Team size impact on design time.

figure

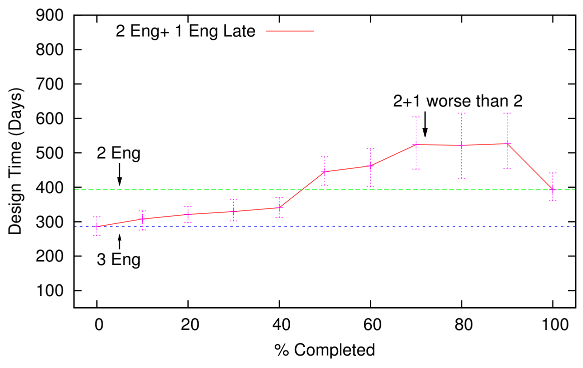

Fig. 7: Brook's Law verification with PUMA. Adding an engineer late to a project makes it later. Eng stands for engineer.

figure

perform 200 simulations for each configuration. The error bar is centered at the median of the design time, the line delimiters the first quantile & the third quantile. To find the COCOMO model parameters, we perform a regression using R [19]. For the project shown in Figure 5 the COCOMO model uses a *b* of 1.19, an very close to the COCOMO embedded projects 1.2 suggested constant. An initial observation from the plot is that the economy of scale shown by μDSim is consistent with software models like COCOMO.

Project Team Impact

Software engineers are aware that compressing a project schedule has a big impact on design effort. Doubling the number of engineers does not reduce the design time by half. The reality is much worse, increasing the number of engineers increases overheads and the project may not have enough independent components for everybody. All the software projects have a point when there is an exponential increase in design effort for any further design time reduction.

Increasing the number of engineers can reduce the design time of a given project. Figure 6 shows the impact on design time as we increase the number of engineers on PUMA. Going from 1 engineer to 2 engineers reduces the design time by 30%. Adding two more engineers further reduces design time to 40%, six engineers reduce the design time by just 13%. Clearly, adding more engineers does not significantly reduce the project completion time. The reason is that PUMA is not complex enough to keep more than 6 engineers working simultaneously on the front end design without significant overheads. Another key observation from Figure 6 is that adding engineers reduces the unpredictability of the project. As the number of engineers is increased we see that the first and third quantiles are closer to the median, thereby reducing the unpredictability.

Brook's Law

Brook's Law [6] states that "adding people to a late project makes it later". To evaluate Brook's law, Figure 7 evaluates the impact of adding 1 additional engineer late to the PUMA project while 3 engineers are involved from the beginning of the project. The two dashed lines are the design times when 2 and 3 engineers join the project from the beginning. The solid line corresponds to the case when 2 engineers start the project and one additional engineer joins later. For the PUMA example, as long as the engineer joins before 40% of the project is completed, it is still possible to provide some design time reduction. Nevertheless, when the additional engineer is added after 50% of the project is completed, the project finishes later. Therefore, this behavior fully complies with observations from Brook's law.

III.D. Sensitivity Analysis

III.D.1 Engineer Model Characterization

μDSim is designed to be as simple as possible while achieving a high correlation with estimations of processor design times. This section quantifies the importance of each of the parameters in the engineer model. After learning is deactivated, we perform regressions with asynchronous and synchronous communication. Each of the respective μDSim simulations has a degraded operation mode with a lower correlation against the reported design times.

The correlation for the default μDSim is 0.98 as shown in Table II The correlation obtained when learning is deactivated from all the modules is 0.84. This shows that we cannot ignore the learning process for engineers during the progress of a project. We study the effects of communication in our model by deactivating asynchronous communication in the simulator, which results in a correlation of 0.90. Deactivating synchronous communication further reduces correlation to 0.89.

III.D.2 Processor Description Characterization

The processor description in Section B makes two assumptions that are verified in this section: Cyclomatic complexity vs lines of code as design effort metrics, and that the algorithm for partitioning the components in a project is fairly insensitive to component size.

Cyclomatic Complexity

Previous work [1] propose the use of the number of HDL statements as a good proxy for design complexity. We use Cyclomatic complexity instead, because it is a metric easier to estimate for computer architecture projects and it is more resilient to component size selection.

TABLE III: Cyclomatic complexity vs number of statements.

table

Module Name	Measured Time	Cyclomatic median	Stmts median
PUMA	240	236	564
IVM	460	536	1103
SCALE	380	424	1019
OpenSPARC	440	454	1230
Victoria Falls	360	396	1428
Niagara 2	720	678	2100
Correlation (τ)	1	0.97	0.89

For the IVM, PUMA, and OpenSPARC projects, we have access to the complete source code. Table III compares the μDSim predictions when Cyclomatic complexity and number of statements is used. The results show Cyclomatic complexity is even better than lines of code. The reason is that as explained in Section B, Cyclomatic complexity is more tolerant different coding styles used across the projects.

We compute the correlation between the design effort and the design times predicted by μDSim both with Cyclomatic complexity and number of statements. A correlation as high as 0.97 indicates that Cyclomatic complexity is more suited for the prediction of design times rather than number of statements.

Component Selection

It is difficult to select a component division that matches the equivalent HDL file that engineers may perform if such design is to be implemented. Using Cyclomatic complexity, implies the feasibility of partitioning blocks, nevertheless, different partitions have different component graphs which can yield to different design times.

To validate the resilience of our approach, we evaluate OpenSPARC. In this case, we join files before accounting for the Cyclomatic complexity. By default, μDSim performs 7840 small tasks for OpenSPARC (design, code, test). We cluster files to double the average task size, this reduces the number of tasks to 4096 but only increases the design time by 3%. This is a small difference as their respective standard deviation is over 10%. Even quadrupling the average task size (2368 tasks) only changes the design time by 4%.

III.E. Architectural Example: Issue Logic

This section showcases the application of μDSim in the evaluation of an architectural proposal. While it is important to estimate the overall processor design time, most research papers only focus on a small subsection of the processor. Here, we evaluate the design time of two issue logic designs: SEED [5] and the original IVM implementation. The proposed SEED window delivers better frequency, power, and area. This favorable behavior under more traditional metrics, seems to imply that SEED is superior to previous issue logic designs. This section shows a step by step method to gather further insight by evaluating the design effort of different competing designs.

Specify SEED Components

The design components are directly obtained from the authors, we use the blocks explained on the SEED paper: Dispatch (Rename Extensions, Dispatch scoreboard and DepTable) and Issue (Issue Buffer and Issue Scoreboard). To estimate the Cyclomatic complexity for both components, we implement the pseudo code for the control for each design and account the number of independent paths. We estimate that the Dispatch and Issue have Cyclomatic complexities of 96 and 88 respectively [4].The SEED paper includes synthesis results, and the authors have disclosed that a it took a single student a full quarter to implement the issue logic described in the paper.

Run μDSim with SEED

The IVM scheduler is a single component with 58 Cyclomatic complexity. To compare SEED vs the original IVM scheduler, we replace the scheduler component by the 2 SEED components (Dispatch and Issue). We perform 200 simulations to obtain the design time distribution for a single engineer. The original IVM ternary is (473, 536 ,620) while an IVM core with a SEED window ternary is (515, 589 ,681). By looking at the median, we can see that SEED window requires 589 days instead of 536 days. This is an overhead of 53 days or close to 3 months which is consistent with the time reported by the authors. We also performed a simulation with 2 engineers. In this case, the design time increased from 384 days to 410 days or 26 days overhead.

In addition to the median design time, the ternary reported by μDSim also includes a risk factor. The wider the difference between first quantile and the third quantile indicates a higher risk. Replacing SEED increases the difference from 66 days to 89 days for the 2 engineer simulation. This means that not only SEED increases the design time by over two months but it also increases the probability of having time overruns.

While the original SEED study includes performance, power, and area estimations, it does not address design time impact. μDSim makes possible to compare across designs, on this example SEED requires 10% additional design time for an Alpha-like processor.

IV. Related Work

To our knowledge μDSim is the first simulator for the quantitative estimation of processor design time. There is little work on the modeling and measurement of design complexity which is a direct actuator on design time. μComplexity [1] introduced a quantitative framework to measure the design effort (complexity) for processors as well as Application Specific Components (ASICs). This work identifies a set of design metrics which are highly correlated with design effort. Most notably, the number of lines of HDL code, and the linear combination of LoCs with the sum of all the fan-ins for the logic cones in a design, are both good estimators of design effort.

The work proposed by Zhang et al [20] measure verification effort by using formal verification methodologies. They quantify the verification effort as the number of reachable states in

[4] Each of these blocks consist of several Cyclomatic complexity nodes.

a design, the testing effort is determined as the product of the length of test vectors and the number of test patterns.

The field of software engineering has produced analytical models like the Constructive Cost Model (COCOMO) [3, 21] which uses a hierarchy of forms to estimate design effort, cost, and schedule of software projects. Most COCOMO models deal with static, single-valued models that compute design effort/cost as a function of the estimated program size. These models can be further refined by including attributes for the development tools, engineering skills, and project characteristics that may act as "cost drivers." Furthermore, the impact of those attributes can also be assessed on each step on the software design process (requirements, design, coding and testing). The SLIM model by Putnam [22] is another relevant estimator for software development effort. This model also estimates design time and effort as a function of project size. Most design effort estimators for software projects share the same problem: they require an accurate estimation of the size in LoCs of the project to be analyzed. The model developed in [10] shows the effects of communication on group productivity in software development, μDSim presents a similar behavior by hardware design teams.

Other studies in software engineering, such as Function Point Analysis (FPA) [4], estimate both the size of the project and its design effort. FPA reflects the amount of functionality that is relevant for a project, independently of its implementation. This analysis is based around a unit of measurement named "function points" (FPs) which estimates the size of an information system. FPA relies on the determination of certain characteristics of the software to be developed (inputs, outputs, operations, files, etc.). By assigning FPs to each aspect of the project and using data from past projects, it is possible to estimate the effort required to implement the calculated FPs for the current project.

Research in industrial engineering and organization has produced a significant amount of results in economic simulation. Delphi methods [23] introduced consensus-based estimation techniques to predict the effort required for industrial and economic projects. Finally, other project completion frameworks rely heavily on Monte Carlo methods [24] to simulate the random nature of many management problems.

V. Conclusions and Future Work

This paper proposes a novel simulation infrastructure to estimate design time. As the evaluation shows, μDSim can estimate the design time for a given processor design in a reliable fashion. The capacity to estimate design time allows designers to avoid possible complexity pitfalls, that are hidden otherwise, early in the development cycle. This leads to shorter development cycles.

μDSim is a simple but accurate design time estimator. We use Cyclomatic complexity, a metric from software engineering, to understand the effort involved in implementing a specific processor element.

Cyclomatic complexity is a key parameter, but it is not enough to estimate the design effort. Decreasing the Cyclomatic complexity by 10% does not yield a 10% design time reduction. Cyclomatic complexity reductions yield different design time reductions dependent on the overall design. μDSim is able to find the design time change for a given Cyclomatic complexity change.

The evaluation shows that μDSim achieves very high correlation levels, 0.98, when predicting the design times for the processor designs found in the testing set. The predictive nature of μDSim allows it to solve some of the shortcomings of other design effort models [1], which require perfect knowledge of the project and have to adjust for the productivity for each project team.

The μDSim ternary allows designers to produce quantitative estimations for the expected design time of their architectural proposals. This allows designers further insight available during the decision process between competing optimizations and architectures. Without quantitative approaches to estimate design times for design proposals, processor architecture lacks a very important dimension of evaluation. For example, some studies aim at reducing processor validation time, by adding hardware support for bug location and tolerance [25, 26]. Using μDSim, designers can estimate the design time associated with the hardware support required to implement these proposals. Thus allowing the architect to determine, whether or not, the expected reduction in validation time is overshadowed by the design time required to implement a given enhancement.

μDSim can be extended to model different factors. For example, μDSim assumes no reuse and it still achieves a very good estimation for the Victoria Falls system which has an extensive reuse from Niagara 2. Nevertheless, we think that μDSim is a good starting point capable of predicting front-end and verification design times for multiple academic and industrial projects. We expect that further additions could add frequency push, backend and reuse models.

VI. Acknowledgements

We would like to thank Jayakumaran Sivagnaname, Jiri Gaisler, Nicholas Wang, Rodrigo Liang, and Krste Asanovic for providing design time information. This work was supported in part by the National Science Foundation under grants 0546819, 0720913, and 0751222; Special Research Grant from the University of California, Santa Cruz; Sun OpenSPARC Center of Excellence at UCSC; gifts from SUN, nVIDIA, Altera, Xilinx, and ChipEDA. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] C. Bazeghi, F.J. Mesa-Martínez, and J. Renau, "μComplexity: Estimating Processor Design Effort," in International Symposium on Microarchitecture, Nov 2005.
[2] D. Weaver, "Personal communication," Sun Microsystems, 2008.
[3] B. Boehm, Software Engineering Economics, Prentice-Hall, 1981.
[4] A. Abran and P. N. Robillard, "Function points analysis: an empirical study of its measurement processes," Software Engineering, IEEE Transactions on, vol. 22, no. 12, pp. 895–910, 1996.
[5] F.J. Mesa-Martínez, M.C. Huang, and J Renau, "SEED: scalable, efficient enforcement of dependences," in PACT, 2006, pp. 254–264.
[6] David Brooks, Vivek Tiwari, and Margaret Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," Proceedings of 27th Int'l Symp. on Computer Architecture, 2000.
[7] A.Mockus and J.D. Herbsleb, "Expertise browser: a quantitative approach to identifying expertise," in ICSE '02: Proceedings of the 24th International Conference on Software Engineering, New York, NY, USA, 2002, pp. 503–512, ACM.
[8] J.Wu, T. C. N. Graham, and P. W. Smith, "A study of collaboration in software design," in ISESE '03: Proceedings of the 2003 International Symposium on Empirical Software Engineering, Washington, DC, USA, 2003, p. 304, IEEE Computer Society.
[9] R Glass, Facts and fallacies of software engineering, Addison-Wesley, Reading, Massachusetts, 2002.
[10] D Simmons, "Communications: a software group productivity dominator," Softw. Eng. J., vol. 6, no. 6, pp. 454–462, 1991.

[11] T. J. McCabe, “A Complexity Measure,” in IEEEE Transactions on Software Engineering, 1976.
[12] A. H. Anderson, G. S. Downs, and G. A. Shaw, “RASSP Benchmark-1 and -2: A Preliminary Assessment,” 1995.
[13] T.R. Halfhill, “Ibm trims power4, addds altivec,” in Microprocessor Report, 2002.
[14] G. Karypis and V. Kumar, “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs,” SIAM J. Sci. Comput., vol. 20, no. 1, pp. 359–392, 1998.
[15] Collett International, “2003 IC/ASIC Design Closure Study,” 2003.
[16] N.J. Wang, J. Quek, T.M. Rafacz, and S.J. Patel, “Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline,” in International Conference on Dependable Systems and Networks. Jun 2004, IEEE Computer Society.
[17] R.Krashinsky, C.Batten, M.Hampton, S.Gerding, B.Pharris, J.Casper, and K.Asanovic, “The vector-thread architecture,” IEEE Micro, vol. 24, no. 6, pp. 84–90, 2004.
[18] J. Cohen, Statistical Power Analysis for the Behavioral Sciences, Lawrence Erlbaum, 1988.
[19] r, R: A language and environment for statistical computing, R Foundation for Statistical Computing, Vienna, Austria, 2005.
[20] M. Zhang, A. Lungu, and D.J. Sorin, “Analyzing Formal Verification and Testing Efforts of Different Fault Tolerance Mechanisms,” in Proceedings of the 2009 24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems. IEEE Computer Society, 2009, pp. 277–285.
[21] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby, “Cost models for future software life cycle processes: COCOMO 2.0,” Annals of Software Engineering, vol. 1, no. 1, pp. 57–94, Dec 1995.
[22] L. H. Putnam, “IEEE Transactions on Software Engineering,” in A General Empirical Solution to the Macro Software Sizing and Estimating Problem, 1978.
[23] H.A. Linstone and M. Turoff, The Delphi Method: Techniques and Applications, Addison-Wesley, 1975.
[24] T. Menzies, Zhihao Chen, J. Hihn, and K. Lum, “Selecting Best Practices for Effort Estimation,” IEEE Transactions on Software Engineering, vol. 32, no. 11, pp. 883–895, 2006.
[25] S. Park and S. Mitra, “Ifra: Instruction footprint recording and analysis for post-silicon bug localization of processors,” in Design Automation Conference, Jun 2008.
[26] K. Constantinides, O. Mutlu, and T. Austin, “Online design bug detection: Rtl analysis, flexible mechanisms, and evaluation,” in 41st Annual International Symposium on Microarchitecture (MICRO-41), Nov 2008.