METRICS: A System Architecture for Design Process Optimization *

Stephen Fenstermaker†, David George†, Andrew B. Kahng,
Stefanus Mantik and Bart Thielges†
UCLA Computer Science Dept., Los Angeles, CA 90095-1596
†OxSigen LLC, San Jose, California
{abk,stefanus}@cs.ucla.edu {fen,dg,bart}@oxsigen.com

Abstract

We describe METRICS, a system to recover design productivity via new infrastructure for *design process optimization*. METRICS seeks to treat system design and implementation as a *science*, rather than an *art*. A key precept is that measuring a design process is a prerequisite to optimizing it and continuously achieving maximum productivity. METRICS (i) unobtrusively gathers characteristics of design artifacts, design process, and communications during the system development effort, and (ii) analyzes and compares that data to analogous data from prior efforts. METRICS infrastructure consists of (i) a standard metrics schema, along with metrics transmittal capabilities embedded directly into EDA tools or into wrappers around tools; (ii) a metrics data warehouse and metrics reports; and (iii) data mining and visualization capabilities for project prediction, tracking, and diagnosis. We give experiences and insights gained from development and deployment of METRICS within a leading SOC design flow.

1 Introduction and Motivations

Advances in semiconductor process technology now allow entire multi-million gate systems to be manufactured on a single chip. At the same time, the ability to fabricate silicon has outpaced the ability to design it. Designers rely on IP reuse and integration to meet turnaround time requirements, and this increases complexity of the design process along such axes as cost, testability, etc. Current design processes are less able to meet project goals, in that more designs miss time-to-market windows and/or end up substantially over budget. The resulting increased implementation risk is detrimental to both the semiconductor and EDA industries.

Our work addresses the fundamental issues of understanding, diagnosing, optimizing, and predicting the system design process. We use the term *design process optimization* (DPO) to refer to the continuous optimization of a design process. In automobile, steel, and even semiconductor manufacturing industries, process optimization is a well-established precept. However, before a process can be optimized on a continuous basis, it must first be *measured*. **In contrast to other industries, today there are no standards or infrastructure for measuring and recording the semiconductor design process.** As a result, today's design processes tend to be temporary solutions, unique to individual projects and created based on the intuition of senior engineers. Such solutions typically last for one project only, while the basic problem of unpredictable design success remains unaddressed. In this regime, a product team cannot quantify inefficiencies in its design process, and subjective opinions are formulated as to why a given project failed or succeeded (e.g., failure may be generically blamed on "CAD tools" or "inexperienced design team"). Two fundamental gaps prevent measurement of the design process:

- **Data to be measured is not available.** Most visibility into EDA tools is via log files that are typically created by R&D developers for their own use; these vary wildly across different vendors and offer little insight into "what the tool is thinking" or what aspects of the input instance were critical to success/failure.[1] *If design process information is not reported, it is not available to be recorded and analyzed. If design process information is not reported with consistent semantics (and, hopefully, consistent syntax), it cannot be effectively used to diagnose a (multi-vendor) design flow.*
- **We do not know all the data that should be measured.** Some metrics of tool performance or design artifacts are "obvious", e.g., number of placeable objects, number of unroutes after detailed routing, maximum negative slack over all timing paths, etc. Other metrics are less obviously useful, e.g., it is not clear whether the number of literals after logic optimization has any relationship to the quality of the resulting netlist from a physical implementation perspective. Finally, some metrics are impossible to discern *a priori*, e.g., perhaps it is the number of years of experience of the RTL designer, or the number of spec changes, that is the best predictor of project success.

We see that customers cannot obtain necessary design process data because EDA vendor tools do not report the data. On the other hand, EDA vendors do not necessarily know which data is useful to report. These gaps prevent designers and project managers from finding and correcting inefficiencies in their design processes. *Regardless of whether the chicken or the egg comes first, measurement infrastructure is a necessary condition for measuring, then improving*. The contributions of our METRICS project include:

- Standard *generic tool metrics*, as well as standard *domain-specific tool metrics* (e.g., for timing optimization or P&R), that the EDA industry can standardize on. Unified naming and semantics of common metrics allow multiple tool vendors to report metrics according to the same conventions.

* Work at UCLA was supported by a grant from Cadence Design Systems, Inc. and by the MARCO Gigascale Silicon Research Center.

[1] For example, it is unreasonable to expect a tool user to understand the meaning of a "Max weighted HCut" value that is reported by one placer but not by any other placers. It is also impossible to map log files of competing tools against each other except at the most basic level (runtime, peak memory, ...). This hampers benchmarking.

- Standard system components such as XML (eXtended Markup Language) based metrics transmitters, an Oracle8i-based data warehouse with a standard metrics schema, and Java implementation of a metrics server. *Our infrastructure enables design process data collection in a "no more log files" regime*: design tools and flow scripts transparently write into the design process data warehouse over the inter/intranet, via METRICS-specific standard XML.
- Examples of useful analyses and reports that have been developed on top of an existing METRICS system implementation. We also discuss a wide variety of data mining, analysis and reporting tools that have yet to be developed.

2 Scope of a METRICS System

From a project management perspective, a METRICS system offers the potential for such results as: (i) accurate resource prediction at any point in the design cycle (up-front estimates for people, time, technology, EDA licenses, IP reuse, etc. and correct go / no-go decisions for projects at the earliest possible point); (ii) accurate project post-mortems where everything is tracked, including tools, flows, users, notes, etc. (optimization of the next project becomes possible based on past results, and no loose data or information is left at project end); (iii) return on investment analysis for design IP, design tools and design flows; (iv) management consoles for monitoring of tools, designs and systems at any point in the project life-cycle; and (v) prevention of wasted resources via out of sync runs and duplication of data or effort.

From a tool developer perspective, benefits include: (i) methodology for continuous tracking data over the entire lifecycle of instrumented tools; (ii) more efficient analysis of realistic data, i.e., the developer can rely on the collected data because it is from real designs (no more extrapolation of tiny artificial "benchmarks", or collection of source files for test cases and re-running in house, is needed); (iii) easier identification of key design metrics and effects on tools, via standardized vocabulary and schemata for design or instance attributes; and (iv) improved apples-to-apples benchmarking (and guidance as to what the apples are in the first place).

In some sense, METRICS extends the "Measure, then Improve" precept to "Collect, Data-Mine, Measure, Diagnose, then Improve". Thus, METRICS is not limited to recording of tools-specific or design instance-specific attributes: it includes other design process-related metrics, such as communication metrics, tool metrics, design artifact metrics, and design flow metrics. For example, basic instrumentation of the design process would record such information as which version of what tool was called on what revision of what block, by which design engineer at what time on what machine, etc. Given this scope of data collection, the possibilities for ensuing design process optimizations are literally unbounded, as illustrated by the following examples.

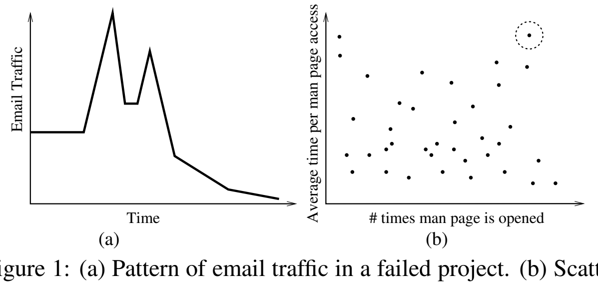

Figure 1: (a) Pattern of email traffic in a failed project. (b) Scatter plot of user accesses to on-line manual pages.

- If a user performs the same operation repeatedly with nearly the same inputs (minutely changing the values of some parameters), the design manager and the tool provider could infer either that the operation is not working as expected (hence, knob-twiddling), or that the quality of the result is unacceptable [14].
- Figure 1(a) shows the pattern of email volume over time in a failed design project.[2] As the deadline nears and a crisis situation sets in, email volume peaks, then drops after the deadline is missed. A second peak occurs in response to a revised deadline, but volume then steadily dwindles after the project fails a second time and team members disengage.
- Figure 1(b) shows a scatter-plot of accesses by tool users to particular pages of on-line documentation. Poor documentation of tools and their capabilities can affect the quality of the design process. A manual page that is opened frequently for long periods of time may indicate poor documentation or an unclear use model for some tool feature.

3 Related Work

We are unaware of any published work on the topic of metrics collection and diagnosis for semiconductor design process optimization. As noted by panelists at the 1998 Design Automation Conference ("Design Productivity: How to Measure It, How to Improve It"), various in-house project tracking systems have been developed (at LSI Logic, IBM ASIC Division, Siemens Semiconductor (Infineon), Sony, Texas Instruments, etc.), each with its own proprietary measures of designs and the design process.[3]

An in-house system used at Texas Instruments since 1975 was discussed in a 1996 DAC Birds of a Feather meeting [3]. This Design Activity System (DAS) collects product metrics as well as license activity information to assess efficiency of tool usage. DAS also provides a standard termination code set for all tools so that designers can more easily diagnose causes of tool failure. The work of [13] proposes to classify designs based on certain metrics (number and complexity of module interfaces, etc.). Software metrics (cf., e.g., the Capability Maturity Model for Software Development [20]) may also be applied, particularly in the front-end design context where system design is very similar to software design. We note that such efforts do not address the above-noted structural and standardization gaps between designer, EDA provider, and process diagnosis/improvement infrastructure.

Additional relevant links include the following.

- **(Web-based) design support** has been proposed for distributed/collaborative design. Distributed web-based design environments include Fujitsu's *IPSymphony* [9], the Berkeley WELD project [23], and the VELA project [22]. A notable project tracking infrastructure is given by the N-dim design support system [6, 19, 17], which collects and maintains design information, and also analyzes this information for project management purposes. N-dim tracks changes so that design information is always relevant and complete; information storage is transparent to possible changes in the underlying design process, so that no "loose data" is lost in translation to new projects. However, as the N-dim system is tightly focused on project data tracking, it does not address the issue of *improving* the design process.
- **E-commerce infrastructure** now includes models for marketing and sales of tools and services over the Internet. E-commerce efforts toward time- or token-based EDA tool use, possibly on external server farms, include Avant!'s EDA Mall [8] (which allows users to buy small sessions of Novas VeriLint), and Synopsys and Lucent FPGA design tools hosted

[2] Note that a well-instrumented design process would provide templates for project-related communications, record the incidence of group meetings, etc.

[3] In addition, consulting firms such as Collett International offer proprietary metrics (e.g., "normalized transistor count") and design productivity analysis services.

by Toolwire [1]. The Synopsys IP Catalyst site sells design IP that is priced based on various design metrics. Although web and server farm infrastructure is conducive to recording of design process metrics, it has so far been directed only toward "metered" tool licensing and IP sales.

- **Continuous process improvement (CPI)** [2] is a methodology that analyzes a (design) process and optimizes it on a continuous basis. CPI is the best known embodiment of the "measure, then improve" precept, and is currently applied in most manufacturing industries. [11] gives a method for identifying the most important metrics for collection, to reduce the complexity of data gathering. Techniques for identifying improvements due to CPI are given by, e.g., [24].

- **Data mining** [15, 18, 10] entails the AI- and statistics-based extraction of predictive information from large databases. In recent years, powerful new technologies have emerged that allow organizations to mine data warehouses for predictions of trends and behaviors, as well as decision support.[4] **Visualization** [16] [12] [4] is necessary for human understanding of correlations and trends.

4 METRICS System Architecture

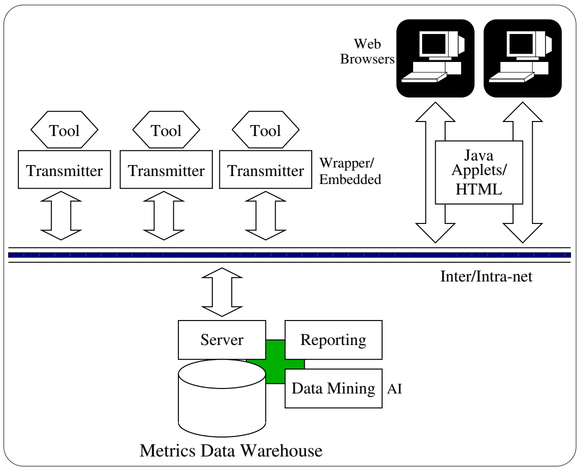

Figure 2: METRICS architecture

The architecture of the METRICS system shown in Figure 2 is a specific implementation of a distributed, client server information gathering system. The EDA tools, which are the data sources, have a thin transmitter client embedded in script wrappers surrounding the tool or actually embedded inside the tool's executable for more flexibility. The tools – which can be located anywhere on an intranet or even the internet – broadcast in real-time as they run using standard network protocols to a centralized server which is attached to a *data warehouse*. The messages transmitted are encoded in industry-standard XML format. Data warehouses are now starting to read, write, and store XML directly (e.g., our current implementation uses Oracle8i), which makes for a more straightforward and robust system. An example of a METRICS message in XML format is shown in Figure 3.

Transmission

Before metrics can be transmitted, they must be collected from design tools. Some aspects of current tools prevent collection of all desired metrics.

[4] A familiar example is the tracking of customer purchases by grocery stores, and mining of this data to optimize shelf placement and promotional efforts.

```
<?xml version="1.0"?>
<!DOCTYPE metrics-doc [
  <!ELEMENT metrics
      (projectID, flowID, toolRunID, name, value)>
  <!ELEMENT projectID #PCDATA>
  <!ELEMENT flowID #PCDATA>
  <!ELEMENT toolRunID #PCDATA>
  <!ELEMENT name #PCDATA>
  <!ELEMENT value #PCDATA> ] >
<metrics-doc>
  <metrics>
    <projectID> 103 </projectID>
    <flowID> 17 </flowID>
    <toolRunID> 223 </toolRunID>
    <name> wirelength </name>
    <value> 1.97e06 </value>
  </metrics>
</metrics-doc>
```

Figure 3: Sample XML for the METRICS system

```
void main(int argc, char* argv[])
{
  PlaceParameters plParams(argc argv);
  Netlist netlist(plParams);
  unsigned PID = grepProjectID(argc, argv);
  unsigned FID = grepFlowID(argc, argv);
  unsigned TID = initToolRun(PID, FID);
  Placer place(netlist, plParams);
  sendMetric(PID, FID, TID, "wirelength", place.getWL());
  ...
  terminateToolRun(PID, FID, TID);
  return 0;
}
```

(a)

```
($File, $PID, $FID) = ARGV;
$TID = '$initToolRun $PID $FID';
open(LOG, "< $File");
while(<LOG>)
{
  if (/\s+Wirelength\s+(\d+).*/)
    { sendMetric($PID, $FID, $TID, "wirelength",$1); }
  ...
}
close LOG;
system "terminateToolRun $PID $FID $TID";
exit 0;
```

(b)

Figure 4: Code examples: (a) API call from inside tools, and (b) wrapper of tool log files

- Currently, log files are the only place from which we can assume metrics are collectable. Not all tools have APIs and extension languages to afford user visibility.

- Tools do not provide enough information in their log files, e.g., metrics from internal loops are not recorded.

- Tools do not have knowledge of the design on which they are working, e.g., a place-and-route tool typically does not know whether it is laying out a state machine or a datapath.

- No project-related metrics are available from tool logs.

- Log files may become corrupted, which can lead to incorrect values of recorded metrics.

To improve metrics collection, we create APIs that can be used by the tool to transfer metrics directly to the server. Tool vendors can insert the APIs into their codes so that as designers use the tools, internal data can be transmitted to the server. APIs also remove the necessity for log files and promote transparency to tool users.

To maintain relations among data sent to the database, each metric is tagged with a 3-tuple of numbers: *project ID*, *flow ID* and *tool run ID*. Project ID identifies the current project on which the tool user is currently working. Flow ID identifies the design flow to which the run belongs. Tool run ID identifies the tool run to which the reported metrics belong. This 3-tuple gives a unique identification for each tool run; sequencing information is maintained by the server.

APIs for the transmitter are as follows:

- **initProject**: Initialize a new project. This function will send a request for a new project ID to the server and set all necessary information for a new project (e.g. environment setup).
- **initFlow**: Initialize a new flow. It is important to distinguish among multiple flows inside a given project. This function will send a request for a new flow ID with a given project ID to the server. Similarly, it will set all necessary information for a new flow. This function requires a valid project ID.
- **initToolRun**: Initialize a new tool run. This function will send a request for a new tool run ID to the server which indicates that a tool is going to be executed. This function requires a valid project ID and flow ID.
- **terminateToolRun**, **terminateFlow**, and **terminateProject**: Terminate tool run, flow or project respectively. Each of these functions has a parameter that is used to pass the termination status of the tool, flow, or project.
- **sendMetrics**: Construct an XML message from the input data and then send the message to the server. This is the core function that transmits data to the server; it must have the ability to recover from transmission failure (e.g., by storing data locally for later transmittal). This function requires two important parameters, the metric name and the value for that metric. The metric name, the corresponding value and identification (project ID, flow ID and tool run ID) are encoded into an XML message similar to that of Figure 3.

If an EDA vendor is unwilling to expose tool metrics or spend developer resources calling our API, we may collect metrics from log files using wrapper scripts. Figure 4 gives example codes for (a) use of APIs inside the tools, and (b) use of a wrapper that gathers data from log files.[5]

Reporting and Data Mining

Reporting and data mining are another important part of the METRICS system, giving project managers and designers more visibility into current or historical projects and designs. In response to user requests, the METRICS system generates various reports and posts these as HTML pages. The current implementation employs web-based reporting which allows access from any place and from any platform. Furthermore, wrappers can be created on top of third-party tools so that data can be transmitted directly to local tools[6], allowing plotting and analysis to be performed locally. Example third-party tools with appropriate interfaces are Microsoft Excel, Lotus Suite, and various data mining tools. In addition to standard HTML generation, Java applets may be used to enhance reporting capability. These applets give users the flexibility to choose different types of graphs, select different sets of data, use different scaling factors, etc. Note that with this approach, data is downloaded to a local machine only once.

[5]One significant feature of the architecture, aimed at ease of deployment and end user use, is the specification of a general purpose network discovery protocol (such as that employed by Jini from Sun Microsystem). This avoids forcing the tools to have a priori knowledge of server network locations. Another feature aimed at increased system robustness is a handshaking protocol between the client and server which allows proper functioning in cases of intermittent and even total network or server failure.

[6]Local tools are the ones that are installed on the client machine.

There are a variety of report types that users can obtain from the METRICS system. These reports can give current status, act as alarms indicating an anomalous condition, or predict future conditions. Below is a table of some high-level reports that have been implemented.

Tool issues	tool aborts runtimes per machine runtimes per IP runtimes per project
Convergence	synthesis-area-speed synthesis-area-power
“IP Mining”	non_tapeout by IP synth_speed by IP convergence by IP IP usage per project
Performance	area vs gates gates vs time violations vs options congestion vs wirelength
Project	Gantt chart of tool usage missed milestones

5 METRICS Schema

When gathering, transmitting, and storing the data from tools to the server, we need to specify a list of metrics (types of data) that need to be transmitted. The metrics can be design instance specific, such as size of chip, wirelength, etc. – or process related, such as type of machine used to run the tools, elapsed time each tool took to complete the run, etc. Most process-related metrics are common across all executions of a given tool, and a number are common across all tools. Figure 5 shows a partial list of such *generic* tool metrics (in our current system, there are approximately 60 generic tool metrics).

peak_memory	peak_disk_kb
peak_cpu_util_%	start_cpu_load
end_cpu_load	machine_type
phys_memory_kb	virt_memory_kb
phys_disk_names	mounted_disk_names
cpu_type	#_of_cpus
host_id	host_name
ip_address	process_exit_status
os_name	os_version
window_mgr_name	window_mgr_version
design_identifiers	start_date_time
end_date_time	runtime_cpu
tool_name	tool_version
tool_vendor	used_command_line_args

Figure 5: Generic metrics for any tools.

There are also many metrics that are tool-specific. Metrics for one type of tool may be different from metrics for another type (e.g., metrics for logic synthesis will not be the same as those for place-and-route). Thus, we categorize tool-specific metrics based on the type of tools that use them. Figure 6 gives example lists of metrics for several tool types. Achieving a standard metrics naming and semantics is a critical METRICS system goal, since it allows portability of the system across tools from different EDA vendors.

6 Current Implementations

A prototype of METRICS has been created by OxSigen LLC for a complete ASIC flow at a major European semiconductor company. A second prototype, more focused on the back end and closely mirroring the architecture discussed above, has been created at UCLA. Figure 7 shows the architecture of the latter, including a mix of Cadence Design Systems place-and-route tools as well as two UCLA

Logic Synthesis	# inputs # registered pins # latches # RAM blocks max clock freq.	# outputs # flip-flops # gates # clock domains # critical paths
Placement	# macro blocks # nets row utilization estimated wirelength	# cells # rows layout area die size
Timing Optimization	max hold violation max setup violation # timing violation min clock cycle	# timing constraints # clock trees # critical paths # repeaters
Routing	# wrong-way routing ave wrong-way WL routed wirelength # routing violation	# cells # nets # layers # vias

Figure 6: Tool-specific metrics for each tool class.

tools. We use wrappers for the Cadence tools and embedded APIs for the UCLA tools. Both prototypes instantiate all parts of a complete METRICS system.

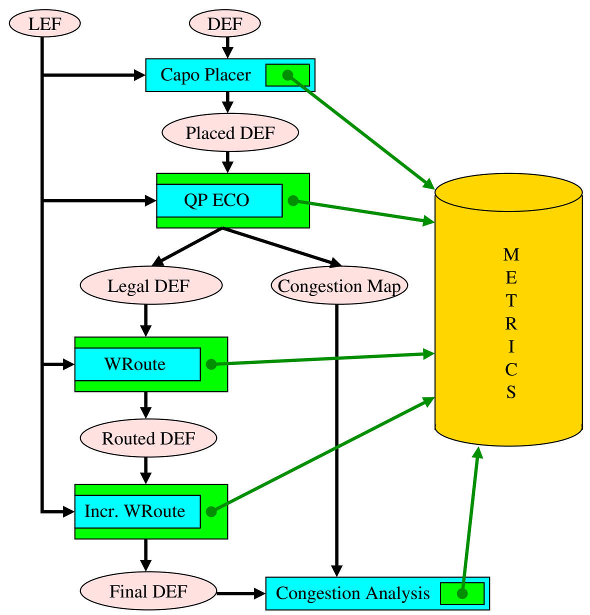

Figure 7: Mixed flow with Cadence/UCLA tools.

Results

Figure 8 is an example report from the OxSigen LLC system, showing the convergence of LVS per block over time. Figure 9 shows the percentage of aborts by design tasks; this view of the design process allows identification of which flow stages are more or less "push-button" than others. Finally, Figure 10 shows a standard correlation report from the UCLA system; here, the plot shows high correlation and a likely direct relationship between number of vias and total wirelength.

We have also performed simple data mining on our METRICS datasets. For example, using *Cubist* [7] we can generate a model that predicts total placement CPU time with average error of approximately 10%. (In the experiment, we train the data mining tool with 56 runs and predict 112 runs. The 168 total runs come from four different test cases. Mean CPU time for all runs is 784.87 seconds, and average absolute error for the prediction of the 112 runs is 84.4 seconds.)

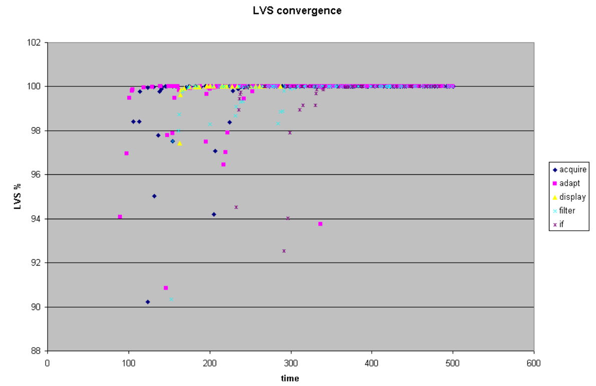

Figure 8: LVS convergence per block.

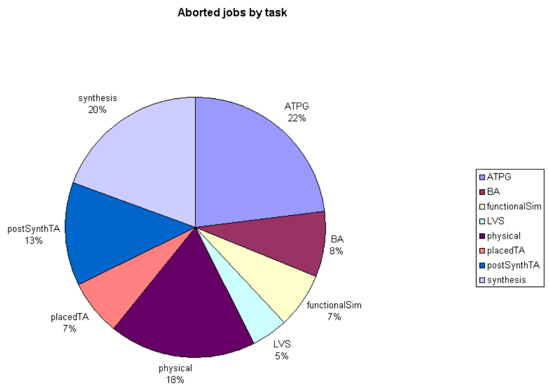

Figure 9: Abort percentage by task.

7 Discussion and Conclusions

A number of issues arise as we go forward with the concept of METRICS. These can be categorized into three types: (i) performance issues, (ii) standardization issues, and (iii) social issues.

Performance Issues

Tool speed should not be degraded significantly with the embedding of METRICS transmitters. Thus, the transmitter should have low overhead and the tool should not become blocked in the event of transmitter failures. **Memory consumption** (memory overhead) should be low.[7] At some point, even if we have selected the best possible configuration for the transmitter, we must still decide the tradeoff between speed, memory, and data availability. E.g., some data may need to be continuously updated whenever values become available, for "status-at-a-glance" reporting. **Security** of transmitted METRICS data must be guaranteed. Transmission should use a secure protocol and encryption.

Standardization Issues

Scope of the METRICS data must be bounded. Not all data are easy to track and capture – e.g., phone traffic, meeting schedules, or drawings on whiteboards. Moreover, not all trackable data is useful, e.g. the number of hours the designer spends browsing the web. **API embedding** in the tool is the best choice for transmitting data, since existing tool logs are such a weak foundation for METRICS; it is also the least intrusive choice. The final goal for our system is indeed "no more log files". There should be a **standardized**

[7] For example, most available memory is reserved for the main process, which is the tool itself. Thus, the transmitter should not request a big chunk of memory that may cause memory swaps and degrade tool performance.

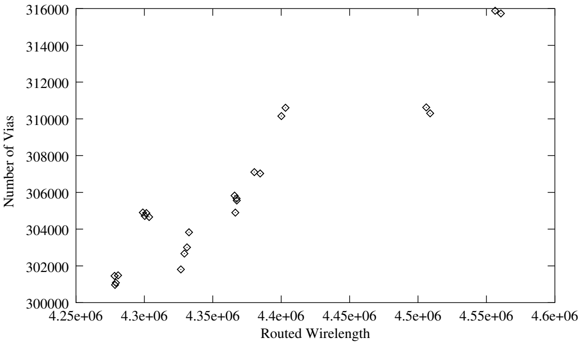

Figure 10: Routed wirelength vs. number of vias.

flow with "Makefile" type build environment for batch chip creation. There is no obvious common way to handle interactive tools yet, so we must be able to metricize flows in a standard way (which requires standard flows). Similarly, there should be **standardized flow terminology** for the CAD and design communities, especially now that so many new hybrid tools are emerging which combine traditional flow steps. Accurate and more worthwhile benchmarking and prediction can be started if there is a standard set of agreed upon milestones ("canonical design states") that occur during the lifecycle of a design. **Standardized data management** (version control) is needed to eliminate lots of custom codes that are used to work around source code control systems in real-world environments. **Project management tools** need to be more standardized and widely used. These tools must act as the metrics transmitters for such project-related information as time allotted for certain tasks. This is critical for prediction of project-level details (e.g., how long to completion from this point, estimation of Gantt charts, etc.).

Social Issues

Monitoring is perhaps the thorniest social issue facing METRICS, since privacy and a non-intrusive design environment are so highly prized. "Big Brother" type issues must be spelled out clearly at the beginning, and buyin from user advocates must be obtained. All data must be anonymized, and any attempt to profile or quantify individual performance on a project is dangerous. **Interpretation of data** is far from straightforward. For example, we might measure the typing activity of two people, one of whom types actively while the other appears idle for hours at a time. The easy inference is that the active typist is more productive. But of course, the person whose keyboard is idle may actually be working on a piece of paper (and thinking how to solve the problem), while the active typist is just typing less well-considered content that will have to be iteratively debugged. In other words, idleness of the keyboard says anything about productivity. Finally, **social metrics** are as important as technical metrics. The ability to improve a design process is highly dependent on the willingness of each designer to stay in the same group, on the documentation and transfer of "best known methods", and even on the level of morale in the organization.

Future Research and Infrastructure

Current efforts are aimed at extending our METRICS system implementation beyond its current project-tracking focus, so that it is more aimed toward true design process optimization. We are also migrating beyond the current internal SOC flow context, so that METRICS is more universally available (e.g., to EDA vendors, other system houses, academic chip design projects, or researchers interested in data mining applications for METRICS). We would like to improve our current infrastructure as follows.

- Polishing of the METRICS infrastructure: (i) more fault-tolerant server design (e.g., via redundancy), (ii) ensuring minimal overhead of the metrics transmission protocol, etc.
- Direct participation by EDA vendors, particularly in the sense of integrating standard APIs for metrics transmission to a standardized data warehouse, and wider participation in standardization of a metrics schema (i.e., expansion of the list of generic and tool class-specific metrics to include additional metrics requested by design teams and/or suggested by tools developers).

Our near-term research objectives include the following.

- Basic research on required data mining technology. Once sufficient metrics have been collected in data warehouses, data mining and visualization techniques will likely be required for design process optimization. This is the most challenging aspect of achieving tool and methodology improvements via instrumentation of the design process; METRICS data collection per se is a necessary but not sufficient precondition.
- Finding sweet spots ("recommended fields of safe or high-quality use") for tools and flows. From these sweet spots, we can categorize tools based on the space of possible inputs.
- Characterizing of a tool's input space (since current real designs are too sparse). Possibly, generation of artificial and/or variant testcase designs, and batch characterization runs, will be required technologies.

References

[1] R. Adhikari, "ASP's Pay-Per-Use Tool", *PlanetIT*, February 2000.
[2] P. S. Adler and K. B. Clark, "Behind the Learning Curve: A Sketch of the Learning Process", *Management Science*, vol. 37 (3), 1991, pp. 267-281.
[3] S. Baeder, Notes from DAC'96 Birds of a Feather meeting, *personal communication*, 2000.
[4] S. S. Bhowmick, S. K. Madria, W. -K. Ng and E. P. Lim, "Data Visualization in a Web Warehouse", *Proc. Adv. in Database Technologies*, 1999, pp. 68-80.
[5] J. S. Busby, "Effective Practices in Design Transfer", *Research in Engineering Design*, vol. 10 (3), 1998, pp. 178-188.
[6] `http://www.ndim.edrc.cmu.edu`
[7] `http://www.rulequest.com/cubist-info.html`
[8] `http://www.edamall.com`
[9] B. Fuller, "Fujitsu Shows Browser-based Design Methodology", *EETimes*, April 1999.
[10] M. L. Gargano and B. G. Raggad, "Data Mining – A Powerful Information Creating Tool", *OCLC Systems & Services*, vol. 15 (2), 1999, pp. 81-90.
[11] A. -P. Hameri and J. Nihtila, "Computerized Product Process: Measurement and Continuous Improvement", *Research in Engineering Design*, vol. 10 (3), 1998, pp. 166-177.
[12] M. Jern, "Information Drill-Down using Web Tools", *Proc. of Visualization in Scientific Computing*, April 1997, pp. 9-20.
[13] M. Keating, "Measuring Design Quality by Measuring Design Complexity", *Proc. International Symposium on Quality Electronic Design*, March 2000.
[14] W. Maly, personal communication, 1998.
[15] C. J. Meneses and G. G. Grinstein, "Categorization and Evaluation of Data Mining Techniques", *Proc. of Intl. Conf. on Data Mining*, Sept. 1998, pp. 53-80.
[16] S. O. Rezende, R. B. T. Oliveira, L. C. M. Felix and C. A. J. Rocha, "Visualization for Knowledge Discovery in Database", *Proc. of Intl. Conf. on Data Mining*, September 1998, pp. 81-95.
[17] J. L. Robertson, E. Subrahmanian, M. E. Thomas and A. W. Westerberg, "Management of the Design Process: The Impact of Information Modeling", available at `http://www.ndim.edrc.cmu.edu/papers/manodespro.pdf`.
[18] M. S. Sousa, M. L. Q. Mattoso and N. F. F. Ebecken, "Data Mining: A Database Perpective", *Proc. of Intl. Conf. on Data Mining*, September 1998, pp. 413-431.
[19] E. Subrahmanian, Y. Reich, S. L. Konda, A. Dutoit, D. Cunningham, R. Patrick, M. E. Thomas and A. W. Westerberg, "The N-Dim Approach to Creating Design Support Systems", *Proc. of ASME Design Technical Conf.*, September 1997.
[20] `http://www.sei.cmu.edu`
[21] D. M. Upton and B. Kim, "Alternative Methods of Learning and Process Improvement in Manufacturing", *Journal of Operations Management*, vol. 16, 1998, pp. 1-20.
[22] `http://www.cbl.ncsu.edu/vela`
[23] `http://www-cad.eecs.berkeley.edu:80/Respep/Research/weld`
[24] W. I. Zangwill and P. B. Kantor, "Toward a Theory of Continuous Improvement and the Learning Curve", *Management Sci.*, 44(7) (1998), pp. 910-920.