Modeling Wire Delay, Area, Power, and Performance in a Simulation Infrastructure

Azmat Hussein and Nicholas P. Carter
Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign
azmat, npcarter@crhc.uiuc.edu

***Abstract*— We present Justice, a set of extensions to the Liberty simulation infrastructure that model area, wire length, and power consumption in processor architectures. Given an architectural specification of a processor, Justice estimates the area of each module of the processor and generates a floorplan of the processor. From the floorplan, Justice computes the length and delay of critical communication paths in the architecture. It then modifies the architectural specification by adding delay elements on communication paths whose delay is one or more clock cycles. This modified architectural description is passed back to the Liberty infrastructure, which creates a simulator for the architecture.**

Justice also estimates the per-access power consumption of each module in an architecture and inserts activity counters to measure how often modules are used. After simulation, a post-processing pass computes the total power consumed by each module and overall power consumption. To illustrate the capabilities of Justice, we simulate a number of VLIW processors and analyze the tradeoffs between power, performance, and wire length in these architectures. In particular, we show that the architectures that have the highest performance when wire length is neglected become some of the worst performers when wire delay is considered and clock rate increased.

I. Introduction

As silicon fabrication technology advances, it becomes more and more difficult to evaluate the performance impact of a change to a processor's architecture without a solid understanding of how that architectural change will affect the physical implementation of the processor. Wire delay is becoming a critical component of overall performance [1] [2], limiting the effectiveness of architectural features that require global communication or unduly increase the distance between other portions of a processor. Similarly, power consumption is becoming a key factor in both portable and desktop systems, affecting both battery life and the cost/feasibility of cooling a processor.

Given the impact that these implementation effects have on the suitability of an architecture for a given application, it is critical that designers consider implementation effects early in the design cycle. In this paper, we present Justice, a set of extensions to the Liberty [3] simulation infrastructure that model the area, power consumption, and critical wire lengths of an architecture. Before simulation starts, Justice parses the architectural description file used by Liberty and estimates the area and power consumption per access of each unit in the architecture. It then generates a floorplan for the architecture and computes the length and wire delay of each communication channel in the architectural description.

If the delay of any communication channel exceeds the clock cycle time of the architecture, Justice inserts FIFO delay queues into the architectural description to model communication time before passing the architectural description to Liberty for simulation. Justice also inserts event counters into Liberty's model of the architecture to record how often each module is accessed during program execution. After simulation completes, Justice multiplies the activity counts for each module by its power per access estimates to compute the total energy and average power used during execution.

The body of this paper begins with an overview of the Liberty simulation infrastructure. We then describe Justice, including how we model power, area, and wire lengths. To evaluate Justice, we present simulation results for a set of VLIW processors that illustrate the impact that wire delay has on performance. Finally, we discuss related and future work, and conclude.

II. Liberty Overview

The Liberty Simulation Environment (LSE) [4] [3] is a toolkit that allows architects to develop high-performance simulators for a wide range of processor architectures. Developed by the Liberty research group at Princeton University [5], Liberty differs from many other simulation environments, such as SimpleScalar [6], in that it does not model any specific processor architecture. Instead, Liberty provides a set of pre-defined components that model common architectural blocks, such as execution pipelines and memory arrays. An architect then defines an architecture as a collection of these pre-defined components, possibly augmented with custom components that model novel aspects of the design.

Liberty defines an architecture as a set of *modules* that represent functional blocks within an architecture. A module consists of a set of *ports* that define its interface to other modules and the code that models the module's behavior. Modules communicate through *channels* that represent communication paths in the architecture. Channels model communication at a somewhat abstract level, using a request/acknowledge protocol to prevent buffer overflows and passing enough context information along with data transfers that changes in the

delay through a channel do not affect the output of simulated programs.

Figure 1 shows an example of a generic cache module that has been defined using lower-level modules. The cache consists of three modules: the tag array, the data array, and the replacement controller, which communicate with each other via the communication channels that are shown as arrows in the figure. When this cache module is instantiated into a design, the user can specify its capacity, associativity, and other parameters, allowing a wide range of caches to be implemented using this generic module.

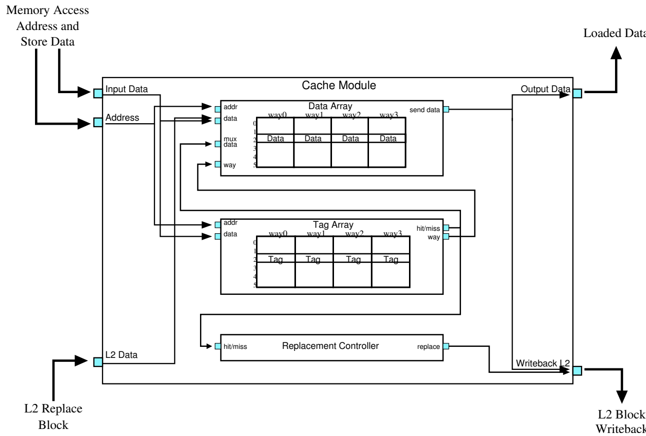

Fig. 1. Hierarchical cache module and connectivity.

To model an architecture in Liberty, a designer writes a configuration file that describes the modules that make up an architecture and the channels that connect them. This configuration file is then passed to Liberty's simulator constructor, *BuildSim*, which generates an executable program binary that simulates the architecture described by the configuration file. Since the simulator that Liberty generates is a customized executable, rather than a generic simulator that interprets a configuration file at run-time to extract simulation parameters, Liberty achieves very high simulation speed, increasing the size and number of the applications that can be used to test an architecture.

Once Liberty has generated a simulator for an architecture, users compile programs to run on the architecture using the IMPACT compiler and a Liberty-specific translator that converts IMPACT's output into Liberty's Xcode format. This compiled program can then be passed as an argument to the simulator binary to simulate the execution of the program and generate high-level simulation results such as the number of cycles required for execution.

Liberty also provides a highly-flexible data collection mechanism that allows users to observe a wide range of simulated events. Using this mechanism, a user defines a custom data collection routine (called a data collector), that becomes part of the architecture's configuration and is invoked whenever the specified event (such as a cache miss, cache hit, or instruction execution) occurs. Since data collection routines are user-specified, they can be as simple or complex as the user requires. For example, a user could define a simple data collector that counted the number of misses in a cache, or a more complex one that sorted cache misses into compulsory, capacity, and conflict misses to give an understanding of why cache misses were occurring in a given program.

We selected Liberty as the basis for Justice because of its flexibility and explicit representation of communication paths. Liberty's "module and channel" representation of processor architectures allows it to model a wide range of architectures, from traditional superscalars and VLIWs to clustered and gridded processors, making it a better match for the needs of researchers exploring solutions to the wire delay problem than a simulator that assumes a particular style of architecture. Similarly, the use of communication channels to pass data between modules allows Justice to identify the important communication paths in an architecture, simplifying the process of floorplanning and wire length calculation. Finally, the semantics of the channel-based communication model allow Justice to insert delays along communication paths that represent long wires without affecting the correctness of the simulation, an important feature for our work.

III. The Justice Extensions to Liberty

Justice works with Liberty and the power models provided by TEM^{2}P^{2}EST [12] to generate power, area, wire length, and performance estimates for an architecture. Figure 2 shows an overview of this process. When Liberty/Justice is invoked, the user's architecture configuration file is passed to Liberty's simulator construction engine, BuildSim, which generates an initial simulator for the architecture. This initial simulator is then passed to Justice's physical approximation generator, which adds power, area, and wire length models to the simulator.

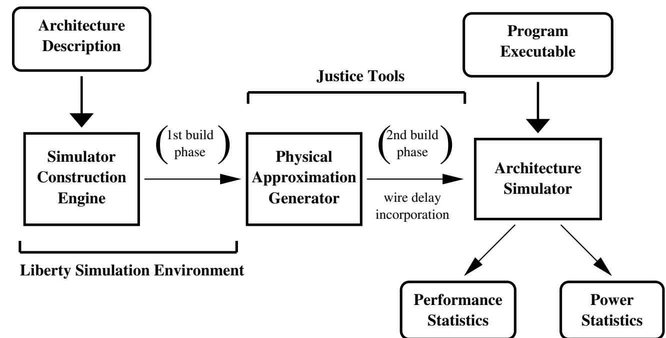

Fig. 2. Basic structure of the simulation system.

Justice generates a per-activity power estimate for each module in an architecture using the power models provided by TEM^{2}P^{2}EST and the parameters of the module. TEM^{2}P^{2}EST supports both analytical and empirical power models. Wherever possible, Justice uses the analytical power models, be-

cause the analytical power models are expected to be more accurate than the empirical models across a wide range of fabrication processes. TEM^2P^2EST's empirical power models are only used on modules, such as ALUs, which it is difficult to model analytically. Once the per-activity power estimate for each module has been generated, Justice adds a data collector routine to the module to record the number of times the module is accessed during simulation.

Justice models the height, width, and total area of each module in an architecture using a combination of analytic and empirical techniques. Wherever possible, architectural structures are modeled as RAM arrays using a model that considers the base area of each RAM cell and the incremental area required as read and write ports are added to the array. When this is not possible, Justice uses empirical models for the area of a module, which were generated by measuring the area of similar units in existing microprocessors and scaling to match the feature size used in the simulation.

Using the area estimates for each module, Justice generates a floorplan for the architecture, treating each module as a fixed block whose aspect ratio cannot be changed, although blocks can be rotated in order to generate a better floorplan. Justice uses a simulated annealing algorithm to optimize the floorplan for a combination of total area and wire length. In future work, we plan to make this optimization criteria accessible to the user, and to allow the user to designate certain communication channels as performance-critical in order to encourage floorplans that minimize the length of those channels.

Once the floorplan is complete, Justice estimates the wire length of each communication channel that connects two or more modules in the architecture as the Manhattan distance between the center of the modules that the channel connects. Justice then estimates the wire delay along each channel based on the properties of the fabrication process being modeled, and divides this delay by the simulated cycle time to determine how many cycles of delay each communication channel will incur, rounding to the nearest cycle.

Based on these wire delay estimates, Justice adds FIFO queues of the appropriate depth to each communication channel whose wire delay is one or more cycles. These queues model the wire delay along the channel, assuming a pipelined design that allows multiple data values to be in flight along a long wire at the same time. The final output of Justice's physical approximation generator is a modified version of the original simulator that augments the original performance model with activity counters, wire delay estimates, and per-activity power estimates for each module in the architecture.

Users run a program on a Liberty/Justice simulation in the same way that they would run a program on a baseline Liberty simulation – by invoking the simulation executable with the program as an argument. The modified simulator executes the program and outputs performance statistics, such as the number of cycles required for execution. When simulation completes, Justice's post-processing phase takes over, multiplying the activity count for each module by the per-activity power estimate for that module to compute the total power consumed in the module.

This combination of pre-processing and post-processing allows Justice to extend the capabilities of Liberty without unduly increasing simulation time and without restricting the types of architectures that can be modeled in Liberty. Justice itself takes very little time to execute, and most of its impact on simulation time comes from the increase in the number of clock cycles that must be simulated when wire delays are taken into account. Justice's modular design also makes it relatively easy to modify its power or area models, for example by replacing the floorplanner with one based on a different algorithm or by changing some of the power models. This allows designers to use Justice effectively even if they feel that one or more of the baseline assumptions that Justice makes are not appropriate for their design.

IV. Experimental Methodology

To illustrate Justice's capabilities, we modeled six VLIW architectures in Liberty/Justice: three conventional VLIW processors, and three clustered VLIW processors. Our baseline VLIW has eight execution units that share a 32-entry register file. Each execution unit can perform both integer and floating-point computations, and two of the execution units can execute memory operations. We refer to this architecture as the 8W2M model because it is eight-wide and can perform two memory operations per cycle. We also model an eight-wide VLIW that can perform four memory operations per cycle (8W4M) and a sixteen-wide VLIW that can perform four memory operations per cycle (16W4M). For our power, area, and wire length studies, we assume that all architectures are fabricated in a 90nm fabrication process.

Our clustered VLIW architectures divide their functional units into two independent clusters, each of which contains half of the functional units and a 16-entry register file. Operations that execute on a given cluster may only access that cluster's register file, although data can be transferred between clusters via explicit move operations. We model three clustered VLIW architectures: an eight-wide processor that can execute one memory reference per cycle from each cluster (8W1M1M), an eight-wide processor that can execute two memory references per cycle from each cluster (8W2M2M) and a sixteen-wide processor that can execute two memory references per cycle in each cluster (16W2M2M).

All of our processor models use 128-KB data and instruction caches. We assume that VLIW instructions are stored in a compressed format, such as the one described in [7], that reduces the amount of space required by NOP operations.

We evaluate these processor architectures using a set of nine benchmark programs taken from the MediaBench suite

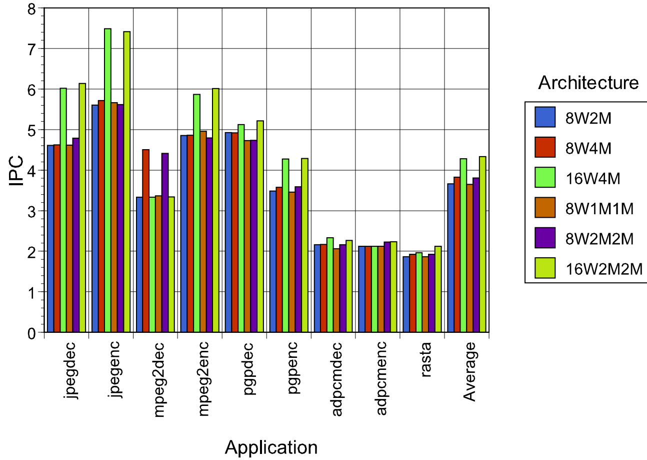

Fig. 3. IPC Without Considering Wire Length

[8]. Where possible, we used both the encoder and decoder portions of each benchmark, although *RASTA*, a speech recognition application, does not have separate encoder and decoder applications. The benchmark programs were compiled for each architecture using the IMPACT compiler [9] to generate high-quality, optimized code.

V. Results

As a baseline for the remainder of this section, Figure 3 shows the average number of instructions executed per cycle (IPC) by our simulated architecture when wire delay and clock rate are not taken into account. Overall IPC numbers are fairly high, as would be expected of highly-optimized media applications, with significant variance between applications. Increasing the number of memory instructions allowed per cycle without changing the architecture's issue width improves IPC by about 4% for both the conventional and clustered VLIW architectures, while increasing issue width from 8 to 16 improves IPC by 8.7% for the conventional VLIW processor and by 10.5% for the clustered processor.

When wire delay and clock rate are not considered, the IPC numbers of clustered and non-clustered processors with the same number of execution units and memory ports are very similar. On some applications, the non-clustered version of an architecture does slightly better than the clustered version because of inter-cluster communication overheads in the clustered version. On others, the clustered version achieves higher IPC than the non-clustered version because long-latency operations in one cluster do not stall computation on the other.

A. Power Consumption

Figure 4 shows Justice's estimate of the average power dissipated by each of our architectures when executing each benchmark. These results assume a 90nm fabrication process and 1.65 V power supply, with all architectures running at 250 MHz, well below the frequency at which wire delays begin to have a significant effect on performance. On average, the eight-wide architectures with four memory ports consume 15% more power than their counterparts with two memory ports, while increasing issue width from 8 to 16 instructions/cycle increases power consumption by 60%. Clustered architectures consume slightly less power than their non-clustered counterparts on almost all applications.

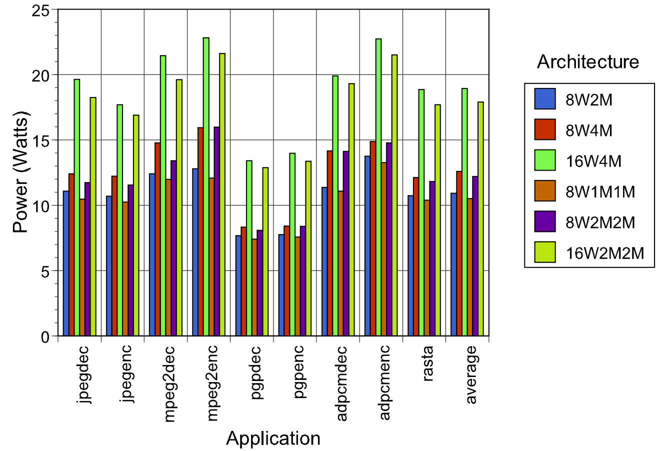

Fig. 4. Power Consumption

To give more insight into these results, Figure 5 shows the contribution of each unit within an architecture to overall power consumption, averaged across all of the benchmarks. Increasing the number of memory operations that an architecture can execute per cycle from two to four increases data cache power consumption by 43%, and also increases the amount of power consumed in queues (including the memory queue), but has relatively little effect on power consumption in other units. Since the data cache consumes about 25% of the power in the base 8W2M processor, these changes explain the overall power differences between architectures with different numbers of memory ports.

The power consumed by the instruction cache is the same for all of the 8-wide architectures and increases by approximately 50% in the 16-wide architectures due to the doubling of the instruction word width. Similarly, power consumption in the execution units varies by only 1% across the different 8-wide architectures, but increases by 55-60% when the issue width is doubled. This sub-linear increase in execution unit power as instruction width increases is due to the fact that execution units consume significantly less power when idle than when executing valid instructions. Since increasing the issue width yields less-than-linear increases in IPC, it is not surprising that it also causes sub-linear increases in power consumption.

B. Area and Wire Lengths

Figure 6 shows Justice's estimates of the chip area required to implement each of our architectures in a 90nm fabrication

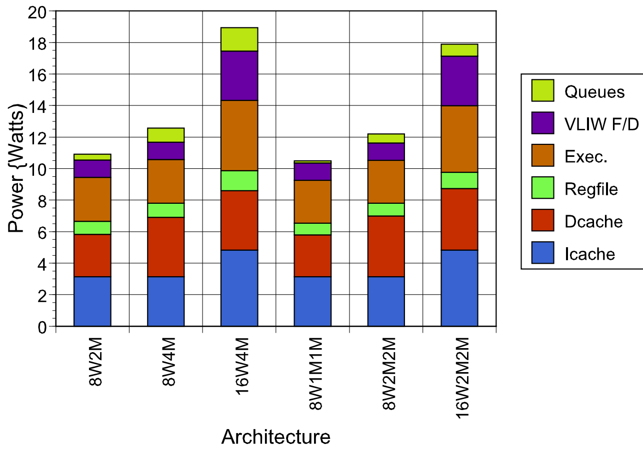

Fig. 5. Breakdown of Power Consumption by Processor Module

process. This graph shows the area of each architecture as the sum of the areas of each of its components, without any blank space that may be introduced during floorplanning. As would be expected, the area taken up by the execution units grows linearly with the number of execution units in the architecture, while the instruction caches of the 16-wide architectures are approximately 20% larger than those of the 8-wide architectures, due to the increase in fetch width in the wider architectures. Similarly, the instruction queues and dispatch logic of the 16-wide architectures are significantly larger than those of the 8-wide architectures.

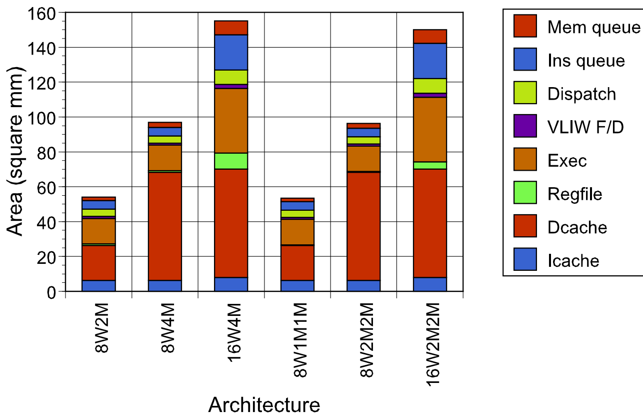

Fig. 6. Area of Each Modeled Architecture

Because the area of an SRAM bit cell increases quadratically with the number of ports, the data caches of the architectures that support four memory accesses/cycle are approximately three times the size of the data caches in architectures that support only two accesses/cycle. Register file size remains constant across the 8-wide non-clustered architectures, but increases when the width of the processor is increased to 16, due to the need to support more ports on the register file. The register files of the clustered architectures are noticeably smaller than those of the non-clustered architectures, as two 16-entry register files are smaller than one 32-entry register file with twice as many ports.

Figure 7 shows the efficiency of the floorplanner in placing each of our architectures, measured as the fraction of the area of the placed chip that is taken up by active circuitry. Most of the architectures achieve between 80% and 95% efficiency, with architectures that contain more units generally having higher efficiency. The 8W2M architecture only achieves 66% placement efficiency, due to an aspect ratio mis-match between its cache and the remainder of its units. This illustrates one weakness of our floorplanner that we intend to rectify in future work – it treats all modules in the architecture as fixed blocks and is unable to alter the aspect ratios of units, such as cache memories, that are more flexible in their design.

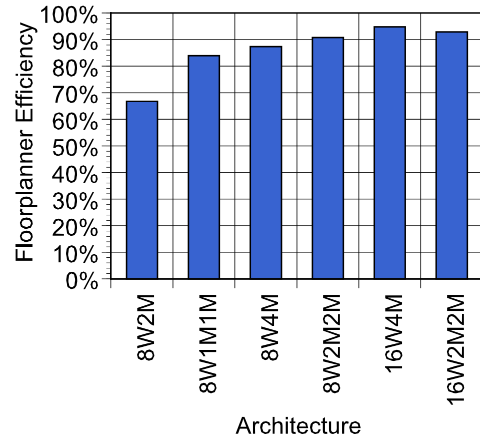

Fig. 7. Efficiency of the Floorplanner

After floorplanning, Justice computed the length of each wiring channel in our architectures, coming up with the data shown in Table I for the longest wire length and delay in each architecture. As would be expected, the 16-wide architectures have significantly longer wires than their 8-wide counterparts, and the clustered architectures have noticeably shorter wires than their non-clustered counterparts. The one surprise in this data is that the longest wire in the 8W4M architecture is only about 10% longer than the longest wire in the 8W2M architecture, in spite of the fact that Figure 6 shows the 8W4M architecture requiring significantly more chip area than the 8W2M architecture. This occurs because the floorplanner does a poor job placing the modules in the 8W2M architecture, leading to a significant amount of blank space in the floorplan. This blank space increases the distance between the centers of the modules in the design, and thus the length of the longest wires.

C. Impact of Wire Delay on Performance

To illustrate Justice's ability to model the effect of wire delay on performance, we simulated each of our target architectures at clock rates ranging from 250MHz to 5 GHz. All architectural parameters other than wire delay, including fabrication

TABLE I

LONGEST WIRE LENGTHS AND DELAYS.

Model	Wire length (mm)	Wire delay (ns)
8W2M	4.379	0.645
8W4M	4.749	0.759
16W4M	6.685	1.505
8W1M1M	4.169	0.585
8W2M2M	3.928	0.521
16W2M2M	5.276	0.937

process, pipeline depth, and memory latencies, were held constant across the clock frequencies. At each clock frequency, Justice automatically inserted FIFO queues of the appropriate depth along the longest wire paths in our architectures to model wire delay.

Figure 8 shows the IPC achieved by each of our architectures as a function of clock frequency, averaged across all of our benchmarks. The leftmost, "ideal" column of the graph shows the IPC each architecture achieves when all wire delays are zero cycles, as would occur at extremely low clock frequencies, while the other columns show IPC at their specified clock rates.

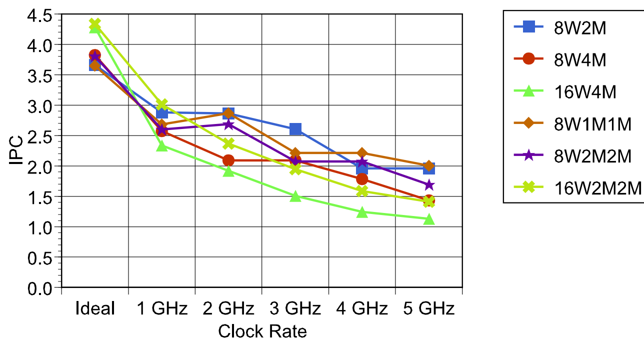

Fig. 8. Impact of Wire Delay on IPC

This graph illustrates the importance of considering wire delay in estimating the performance of processors. While the 16-wide architectures achieve the highest IPC when wire delay is neglected, they quickly become some of the worst performers when wire delay is considered and the clock rate increased. In general, the IPC of an architecture monotonically decreases as clock rate increases, although some artifacts are introduced because Justice rounds the delay on each wire to the nearest full cycle.

To illustrate the effect of wire delay on absolute performance, Figure 9 shows the performance of each of our architectures (measured as the product of IPC and clock rate) relative to the performance of the 8W2M architecture at 250 MHz, a clock rate at which none of the architectures see any wire delay effects in our model. In this graph, we can see that the clustered architectures achieve significantly better performance than their unclustered counterparts as clock rates increase, with the 8W1M1M architecture achieving the highest performance at extremely-high clock rates.

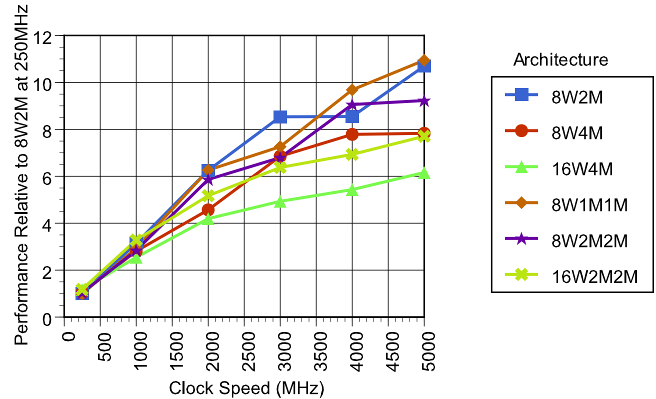

Fig. 9. Impact of Wire Delay on Performance

These results show how Justice can help designers understand the impact that implementation effects will have on their architectures early in the design process, when major changes to an architecture are still feasible. Simulation tools that ignore wire delays can lead designers to choose extremely-complex architectures that are difficult to implement at high clock rates, while failing to estimate power consumption can lead to designs that require expensive heat sinks and/or have poor battery life. By exposing these factors to the designer at the start of the design process, Justice allows designers to explore a wide range of architectures to select the one best suited to their application.

VI. RELATED WORK

Justice builds on the work of a number of efforts that have studied power modeling and/or wire effects in architectural simulators. SimplePower [10], Wattch [11], and TEM^{2}P^{2}EST [12] are three examples of tools that have added power modeling to architectural simulations. Each of these tools uses SimpleScalar [6] as its architectural performance simulator, and augments SimpleScalar with extensions that model the power consumption of different units within an architecture.

While these three tools are all based on SimpleScalar, they use very different approaches to estimate the power consumed by an architecture. SimplePower relies on an empirically-derived table of effective capacitances for each unit in the processor to calculate the total capacitance being charged and discharged on each cycle, and thus the power consumed during the cycle. This approach allows extremely-accurate modeling of the power consumed by a particular architecture, particularly one for which accurate layout-based capacitances are available, but the effort required to generate the capacitance table for each new architecture limits SimplePower's portability.

Watch, on the other hand, uses parameterizable analytic models for the power consumed by each unit in an architecture, allowing it to model a wide range of superscalar architectures.

However, Wattch is strongly tied to SimpleScalar, which assumes a superscalar architecture, and thus would be difficult to apply to a simulator that used a different architectural model, such as a clustered VLIW.

While TEM^{2}P^{2}EST is also based on SimpleScalar, it has a very modular design that makes it easy to apply to other simulation infrastructures. TEM^{2}P^{2}EST models a processor as a collection of functional blocks (FUBs), and estimates the power consumed in each block based on its knowledge of the block and the activity counts generated during simulation. This made it relatively easy to integrate TEM^{2}P^{2}EST's power models into Justice by associating each module in a Liberty simulation with one or more FUBs and augmenting Liberty with data collectors to generate activity counts for each module. TEM^{2}P^{2}EST's chief limitation is that it relies on empirical power density estimates for some of its power calculations, which may not be applicable to architectures and fabrication processes that differ significantly from those used to generate the estimates. While this is an important issue, and one that needs to be taken into account, we felt that TEM^{2}P^{2}EST's modularity and flexibility made it the best starting point for our power models.

Unlike other architectural power models, HotSpot [13] addresses thermal issues in microprocessors by estimating both the rate at which power is consumed in each module on a die and the rate at which heat flows between modules and/or out of the die. To do this, HotSpot generates an estimated floorplan for the system and an "equivalent circuit" that describes the major units in the architecture. It then estimates the rate at which heat is injected into and removed from each module in the architecture in order to estimate the temperature of different portions of the chip during program execution. Given the impact that temperature has on circuit performance, understanding how architectural changes will affect the operating temperature of each part of a chip is extremely important, and we hope to integrate thermal modeling into future versions of Justice so that designers can trade off the effects that architectural changes have on wire length and temperature.

Unfortunately, the previous work on interconnect modeling in computer architectures is significantly more limited than work on power modeling, as wire-related effects have typically been considered only during the circuit-level design and layout of a processor. One existing interconnect modeling tool is the Raphael package from Synopsys, a collection of solvers that extract wire resistances and capacitances from circuit-level specifications [14] to model the electrical and thermal effects of on-chip interconnects. This approach gives very accurate estimates of a circuit's power consumption, but is difficult to apply until late in the design process, making it less suitable for architecture-level tools.

VII. Future Work

The current Justice system has two significant limitations that we plan to address in future work. The first limitation is that Justice relies on the user to provide delay estimates for memory arrays instead of calculating their delays internally. This can lead to inaccurate performance estimates for an architecture if the user does not provide accurate delay parameters for each array in an architecture, and complicates the process of modeling an architecture's performance in different fabrication processes. To address this, we plan to integrate a memory delay model into Justice, which will calculate the access times of each memory array in an architecture and insert them into the architecture model, similar to the way the current wire delay estimator annotates communication channels with delay estimates.

Justice's second limitation is that the floorplanner often generates layouts with some amount of blank space, leading to overestimates of the amount of chip area required by an architecture. These overestimates occur because Justice's floorplanner treats all of the modules in a design as "hard" blocks whose height and width are specified by the area estimator and cannot be changed, although modules may be rotated if this improves the floorplan. While this assumption is reasonable for some modules in an architecture, others, in particular memory arrays, can be laid out in different aspect ratios without significantly affecting their performance. In future work, we plan to extend the floorplanner to support modules with flexible aspect ratios so that users can indicate which of the modules in their architectures can be resized if necessary to improve the quality of the floorplan.

VIII. Conclusion

This paper has described Justice, a set of extensions to the Liberty Simulation Environment that allow designers to estimate an architecture's power consumption, area, and the impact that wire delays will have on performance. Liberty's "module and channel" model of computer architectures exposes important communication paths to the simulator, making it easy to identify wires whose length and delay need to be modeled. In addition, Liberty is extremely flexible, allowing a much wider range of architectures to be modeled than many other simulation tools.

Justice operates in a two-phase manner. Before simulation starts, it generates an area estimate for each module in the design and a floorplan for the architecture. Using this floorplan, it estimates the length of the wires represented by communication channels in the model, and adds FIFO delay queues to channels whose wire delays are more than one clock cycle long. After simulation completes, Justice multiplies the activity counts for each module in the simulation by its per-access power estimates for the module to compute the total power consumed in each module and thus in the entire processor.

Our initial studies using Justice demonstrate the importance of considering power and wire delay in architectural simulation. While the performance impact (ignoring wire delay)

of doubling the issue width of a VLIW processor and of increasing the number of memory references that the processor can perform per cycle are similar, our power results show that adding memory ports to the design has a significantly lower impact on power consumption. When wire delay and clock rate are taken into account, the benefits of clustering a VLIW architecture become clear, as our clustered architectures significantly outperform their non-clustered counterparts. In addition, the benefits of widening the architecture are called into question. At high clock rates, our 16-wide architectures achieve lower overall performance than our 8-wide architectures, due to the increase in wire length between the execution units and centralized control and memory logic.

While these simulations are somewhat idealized, they clearly illustrate the importance of modeling technology effects when evaluating potential changes to a computer architecture. As wire delay and power consumption become more and more significant with advances in fabrication technology, we believe that simulation tools like Justice, which enable designers to easily model technology effects in their simulations, will become indespensible tools for computer architects.

ACKNOWLEDGMENT

This work was supported by the Semiconductor Research Corporation under contract 785. The material presented here represents the conclusions of the authors and does not necessarily represent the opinions of the SRC. We would like to thank David August and the members of the Liberty research group at Princeton University for their help in developing these extensions to Liberty and the anonymous reviewers for their comments on the initial version of this paper.

REFERENCES

[1] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, "Clock rate versus IPC: The end of the road for conventional microarchitectures," in *Proceedings of the 27th Annual International Symposium on Computer Architecture*, Vancouver, British Columbia, Canada, 2000, pp. 248–259. [Online]. Available: citeseer.nj.nec.com/article/agarwal00clock.html

[2] R. Ho, K. W. Mai, and M. A. Horowitz, "The future of wires," in *Proceedings of the IEEE*, vol. 89, April 1991, pp. 490–504.

[3] M. Vachharajani, N. Vachharajani, D. A. Penry, J. A. Blome, and D. I. August, "Microarchitectural exploration with Liberty," in *Proceedings of the 35th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-35)*, November 2002, pp. 271–282.

[4] D. I. August, M. Vachharajani, N. Vachharajani, D. A. Penry, J. A. Blome, and R. Rangan, "Architectural exploration with Liberty," Tutorial presentation at 34th Annual International Symposium on Microarchitecture, Dec. 2001, http://liberty.cs.princeton.edu/.

[5] D. I. August, "Liberty Computer Architecture Research group," http://liberty.cs.princeton.edu/, Jan. 2004.

[6] D. Burger and T. M. Austin, "The SimpleScalar tool set, version 2.0," University of Wisconsin-Madison Computer Science Department, Tech. Rep. 1342, June 1997.

[7] T. M. Conte, S. Banerjia, S. Y. Larin, K. N. Menezes, and S. W. Sathaye, "Instruction fetch mechanisms for VLIW architectures with compressed encodings," in *Proceedings of the 29th Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE Computer Society, 1996, pp. 201–211.

[8] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communications systems," in *Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-30)*, 1997, pp. 330–335.

[9] W. W. Hwu, D. Gallagher, S. Mahlke, D. Lavery, G. Haab, J. Gyllenhaal, and D. I. August, "Compiler technology for future microprocessors," *Proceedings of the IEEE*, vol. 83, no. 12, pp. 1625–1640, Dec. 1995.

[10] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The design and use of SimplePower: A cycle-accurate energy estimation tool," in *Proceedings of Asia and South Pacific Design and Automation Conference (ASP-DAC 2000)*, 2000, pp. 340–345.

[11] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *Proceedings of the 27th International Symposium on Computer Architecture*, June 2000, pp. 83–94.

[12] A. Dhodapkar, C. H. Lim, G. Cai, and W. R. Daasch, "TEM^{2}P^{2}EST: A thermal enabled multi-model power/performance estimator," in *Proceedings of the International Workshop on Power-Aware Computer Systems (PACS),* in conjunction with the *34th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-34)*, 2001, pp. 112–125.

[13] K. Skadron, M. Stan, M. Barcella, A. Dwarka, W. Huang, Y. Li, Y. Ma, A. Naidu, D. Parikh, P. Re, S. Velusamy, H. Zhang, and Y. Zhang, "HotSpot: Techniques for modeling thermal effects at the processor-architecture level," in *Proceedings of the 8th International Workshop on Thermal Investigations of IC's and Systems (THERMINICS-8)*, Oct. 2002, pp. 28–31.

[14] Avant! Technical Staff, *Raphael: Interconnect Analysis Software Product Brochure*, Avant! Corporation, 1998.