McPAT 1.0: An Integrated Power, Area, and Timing Modeling Framework for Multicore Architectures *

Sheng Li[†‡], Jung Ho Ahn[§‡], Jay B. Brockman[†], Norman P. Jouppi[‡]
[†]University of Notre Dame, [‡]Hewlett-Packard Labs, [§]Seoul National University
[†]{sli2, jbb}@nd.edu, [§]gajh@snu.ac.kr, [‡]norm.jouppi@hp.com

Abstract

This paper introduces McPAT 1.0, an integrated power, area, and timing modeling framework that supports comprehensive design space exploration for multicore and manycore processor configurations ranging from 90nm to 22nm and beyond. At the microarchitectural level, McPAT includes models for the fundamental components of a chip multiprocessor, including in-order and out-of-order processor cores, networks-on-chip, shared caches, integrated memory controllers, and multiple-domain clocking. At the circuit and technology levels, McPAT supports critical-path timing modeling, area modeling, and dynamic, short-circuit, and leakage power modeling for each of the device types forecast in the ITRS roadmap including bulk CMOS, SOI, and double-gate transistors. McPAT has a flexible XML interface to facilitate its use with many performance simulators.

*Currently 1.0 beta version is released.

Contents

1 Introduction

Power dissipation, and the resulting heat issues, have become possibly the most critical design constraint of modern and future processors. This concern only grows as the semiconductor industry continues to provide more transistors per chip in pace with Moore's Law. Industry has already shifted gears to deploy architectures with multiple cores [31, 45], multiple threads [25, 28], and large last-level caches [31, 45] so that processors can be clocked at a lower frequency and burn less power, while still getting better overall performance. Controlling power and temperature in future multi-core and many-core processors will require even more novel architectural approaches.

Area remains one of the key design constraints to keep the cost of designs under control because die costs are proportional to the second power of the area [18]. At very small feature sizes, little margin exists between design rules and manufacturing process variations, leading to an average 5% decrease in expected die yield with each successive technology node for mature IC designs [49]. Therefore, on-chip resources including cores and interconnects must be carefully designed to achieve good trade-offs between performance and cost.

Power, area, and timing need to be studied together more than ever as technology keeps scaling down. However, our ability to propose, design, and evaluate new architectures for this purpose will ultimately be limited by the quality of tools used to measure the effects of these changes. Accurately modeling these effects also becomes more difficult as we push the limits of technology. Future multi/manycore designs drive the need for new tools to address changes in architecture and technology. This includes the need to accurately model multicore and manycore architectures, the need to evaluate power, area, and timing simultaneously, the need to accurately model all sources of power dissipation, and the need to accurately scale circuit models into deep-submicron technologies.

This report introduces a new power, area, and timing modeling framework called McPAT (**M**ulti**c**ore **P**ower, **A**rea, and **T**iming), which addresses these challenges. McPAT advances the state-of-the-art of processor modeling in several directions. First, McPAT is an *integrated* power, area, and timing framework that enables architects to use new metrics combining performance with both power and area such as energy-delay-area product (EDAP), which are useful to quantify the cost of new architectural ideas. McPAT specifies the low-level design parameters of regular components (interconnects, caches, other array-based structures, etc.) based on high-level constraints (clock rate and optimization target) given by a user, ensuring that the user is always modeling a reasonable design. This approach enables the user, if they choose, to ignore many of the low-level details of the components being modeled.

Second, McPAT models not just dynamic power but also static and short-circuit power. This is critical in deep-submicron technologies since static power has become comparable to dynamic power [27, 50]. By modeling all three types of power dissipation, McPAT gives a complete view of the power envelope of multicore processors.

Third, McPAT provides a comprehensive solution for multithreaded and multicore/manycore processor power. Contemporary multicores are complex systems of cores, caches, interconnects, memory controllers, multiple-domain clocking, etc. McPAT models the power of the important components of multicore processors, including all the components listed above. McPAT supports detailed and realistic models that are based on existing OOO (out-of-order) processors. McPAT can model both a reservation-station-model and a physical-register-file model based on real architectures, including the Intel P6 [22] and Netburst [19].

Fourth, McPAT handles technologies that can no longer be modeled by linear scaling assumptions. The simple linear scaling principles are no longer valid because device scaling has become highly non-linear in the deep-submicron era. McPAT provides an integrated solution that models all the power sources. Our power-modeling tool makes use of technology projections from ITRS [50] for dynamic, static, and short-circuit power; as a result, this tool will naturally evolve with ITRS even beyond the end of the current road map.

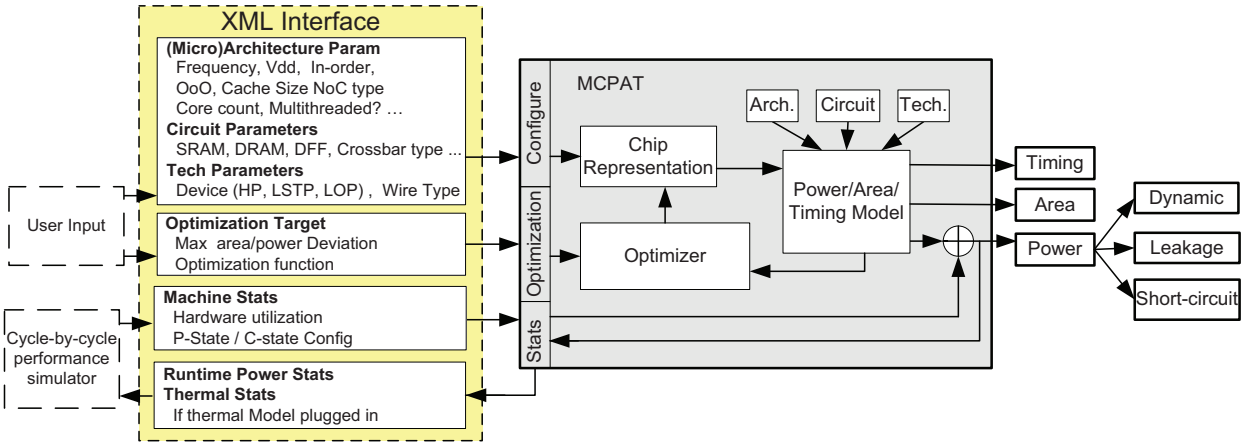


Figure 1: Block diagram of the McPAT framework.

2 McPAT: Overview and Operation

McPAT is the first *integrated* power, area, and timing modeling framework for multithreaded and multicore/manycore processors. It is designed to work with a variety of processor performance simulators (and thermal simulators, etc.) over a large range of technology generations. McPAT allows a user to specify low-level configuration details. It also provides default values when the user decides to specify only high-level architectural parameters.

Figure 1 is a block diagram of the McPAT framework. Rather than being hardwired to a particular simulator, McPAT uses an XML-based interface with the performance simulator. McPAT uses an XML parser [23] developed by Berghen et.al to parse the large XML interface file. This interface allows both the specification of the static microarchitecture configuration parameters and the passing of dynamic activity statistics generated by the performance simulator. McPAT can also send runtime power dissipation results back to the performance simulator through the XML-based interface, so that the performance simulator can react to power or even temperature data. This approach makes McPAT very flexible and easily ported to other performance simulators. Since McPAT provides complete hierarchical models from the architecture to technology level, the XML interface also contains circuit implementation style and technology parameters that are specific to a particular target processor. Examples are array types, crossbar types, and CMOS technology generations with associated voltage and device types.

The key components of McPAT are (1) the hierarchical power, area, and timing models, (2) the optimizer for determining circuit level implementations, and (3) the internal chip representation that drives the analysis of power, area, and timing. Most of the parameters in the internal chip representation, such as cache capacity and core issue width, are directly set by the input parameters.

McPAT's hierarchical structure allows it to model structures at a low level including underlying device technology, and yet still allows an architect to focus on a high-level architectural configuration. The optimizer determines missing parameters in the internal chip representation. McPAT's optimizer focuses on two major regular structures: interconnects and arrays. For example, the user can specify the frequency and bisection bandwidth of on-chip interconnects or the capacity, associativity, the number of cache banks, while letting the tool determine the implementation details such as the choice of metal planes, the effective signal wiring pitch for the interconnect, or the length of wordlines and bitlines of the cache bank. These optimizations lessen the burden on the architect to figure out every detail, and significantly lowers the learning curve to use the tool. Users always have the flexibility to turn off these features and set the circuit-level implementation parameters by themselves.

The optimizer generates the final chip representation, which is used to compute the final area, timing, and peak power. The peak power of individual units and the machine utilization statistics (activity factor) are used to calculate the final runtime power dissipation.

The detailed work flow of McPAT has two phases: the initialization phase and the computation phase. Specifically, in order to start the initialization phase a user first specifies static configurations, including parameters at all three levels, namely, architectural, circuit, and technology level. Architectural level parameters are similar to the parameters used in the configuration files of performance simulators, including the number of cores, the number of routers, shared last-level cache parameters, core issue width, OOO renaming schemes, OOO scheduling schemes, the number of hardware threads, and so on. Since McPAT needs to be paired up with a performance simulator for computing runtime power, a user can write a simple script to extract these architectural parameters from the performance simulator configuration files and generate the XML interface file for McPAT. Circuit-level parameters specify implementation details. For example, one can specify a certain array to use flip-flop based cells rather than SRAM based cells or to use a double-pumped crossbar for on-chip routers. Technology-level parameters include device type (high performance, low standby power, low operating power [50]), and interconnect. The static inputs also include the optimization options, such as the maximum area deviation, the maximum power deviation, and the optimization function. Once all the static configurations are set, the initialization of McPAT can be called by the performance simulator at the beginning of simulation.

During the initialization phase, McPAT will generate the internal chip representation using the configurations set by the user. The main focus of McPAT is accurate power and area modeling, and the target clock rate is actually used as a design constraint. Local greedy optimizations are used, except when there are obvious needs for considering the interplay of multiple components, which means McPAT finds the best solution for each component and assumes they will come up with the best global configuration when individual components are put together. The optimization space that McPAT explores can be huge especially when there are many unspecified parameters. McPAT does intelligent and extensive search of the design space. McPAT optimizes each component by varying circuit implementations. For example, McPAT varies the sub-array implementations of SRAM-based array components and metal layer/wiring pitch of interconnects. For each architectural component, McPAT first finds valid configurations meeting timing constraints. Then, if the resulting power and/or area are not within the allowed maximum deviation of the best value found so far, the configuration is discarded. Finally, McPAT applies a user-specified optimization function to find the one with the highest score among the configurations satisfying the power and area deviation. Users can easily add new objectives such as leakage power and assign different weights for each. During this initialization phase, the internal chip representation that meets the target clock rate is formed, and the area of the target chip is reported. Dynamic energy per access and leakage power of each individual component are obtained.

Although local greedy optimizations are used for most components, there are exceptions when components need to be considered together. Currently, McPAT considers three global optimizations: 1) Bypass logic is assumed to be routed over functional units, register files, and reservation stations. 2) Global interconnects (links between routers) are assumed to be routed over last-level caches, if they present, in horizontal, vertical, or even both directions. 3) Clock distribution networks are assumed to cover the whole chip for a global H-tree and whole domains such as cores for semi-global H-trees. Local clock distribution networks are assumed to cover each entire domain. To accurately model the interplay of the scenarios, global optimization should be applied. For example, if the height of the interconnects is no larger than that of the L2 cache, McPAT assumes the interconnect can be routed over the last-level cache horizontally.

The computation phase of McPAT is called by the performance simulator during simulation to generate runtime power numbers. Before calling McPAT to compute runtime power numbers, the performance simulator needs to pass

the statistics, namely, the activity factors of each individual components to McPAT via the XML interface file. As shown in Equation 1, the activity factor of a component is the product of access count of the component and the average hamming distance of all accesses for the time interval. By changing the time interval, one can change the granularity of runtime power computation. If the performance simulator calls McPAT for runtime power computation every cycle, a cycle accurate power consumption profile will be generated, which will be useful to study realtime power spikes. If the performance simulator calls McPAT for runtime power computation after power simulation is complete, an averaged power profile will be generated.

$$ActivityFactor = (AccessCount * ((\sum_{i=1}^{n} HammingDistance)/n))/n \quad (1)$$

In Equation 1, n is the cycle count for a given simulation period, *AccessCount* is the number of accesses to a specific component during the period, and the $HammingDistance$ is the total number of flipped bits for two consecutive accesses. If the performance simulator cannot track the Hamming Distance, McPAT assumes that all bits are flipped per cycle. It is always best for the performance simulator can provide the activity factor for each individual component, however, McPAT also has the ability to reason about activity factors for components as long as the basic statistics information is provided by the performance simulator. For example, if the performance simulator can only track the number of memory instructions rather than the detailed information of activity factors of the load or store queue, McPAT assumes that each memory instruction will involve one read, one write, and one or two search operations (depends on the hardware specification) on load and store queues. This assumption is made based on the default implementations of the load and store unit as discussed later in this report.

McPAT runs separately from a performance simulator and only reads performance statistics from it — therefore, its impact on the simulation speed of the native performance simulator is minimal. Although the initialization phase of McPAT may take some time to complete because of the huge search space, it will not affect the simulator speed significantly since it needs to be done only once at the beginning of the simulation. During the computation phase, some simulator overhead may result from added performance counters.

3 Integrated and Hierarchical Modeling Framework

In order to model the power, area, and timing of a multicore processor, McPAT takes an *integrated* and *hierarchical* approach. It is integrated in that McPAT models power, area, and timing *simultaneously*. Because of this McPAT is able to ensure that the results are mutually consistent from an electrical standpoint. It is hierarchical in that it decomposes the models into three levels: architectural, circuit, and technology level. This provides users with the flexibility to model a broad range of possible multicore configurations across multiple implementation technologies. Taken together, this integrated and hierarchical approach enables the user to paint a comprehensive picture of a design space, exploring tradeoffs between design and technology choices in terms of power, area, and timing.

3.1 Power Modeling

As shown in Equation (2), power dissipation of CMOS circuits has three main components: dynamic, short-circuit, and leakage power. All three contribute significantly to the total power dissipation of multicore processors fabricated using a deep-submicron technology.

$$P_{total} = \underbrace{\alpha C V_{dd} \Delta V f_{clk}}_{Dynamic} + \underbrace{V_{dd} I_{short\_circuit}}_{Short\_circuit} + \underbrace{V_{dd} I_{leakage}}_{Leakage} \tag{2}$$

The first term is the *dynamic power* that is spent in charging and discharging the capacitive loads when the circuit switches state, where C is the total load capacitance, V_{dd} is the supply voltage, ΔV is the voltage swing during switching, and f_{clk} is the clock frequency. C depends on the circuit design and layout of each IC component; we calculate it using analytic models for regular structures such as memory arrays and wires, along with empirical models for random logic structures such as ALUs. The activity factor α indicates the fraction of total circuit capacitance being charged during a clock cycle. We calculate α using access statistics from architectural simulation together with circuit properties.

The second term is the *short-circuit power* that is consumed when both the pull-up and pull-down devices in a CMOS circuit are partially on for a small, but finite amount of time. Short-circuit power is about 10% of the total dynamic power, however it has been reported that the short-circuit power can be approximately 25% of the dynamic power in some cases [42, 62]. When circuit blocks switch, they consume both dynamic and short-circuit power. The inherent circuit property determines the ratio of the short-circuit power to the dynamic power, which is a strong function of the V_{dd} to V_{th} ratio. Since the V_{dd} to V_{th} ratio shrinks for future low power designs, short-circuit power is expected to become more significant in future designs that require lower power and longer battery life. We compute the short-circuit energy per switch of a gate, using the Equations 3 to 5 derived in [42].

$$E_S = \frac{1}{\frac{1}{E_S(t_T << \tau)} + \frac{1}{E_S(t_T >> \tau)}} \tag{3}$$

$$E_S(t_T << \tau) = \frac{3}{10} * \frac{(0.5 - v_T)^3}{\alpha^2 2^{3v_T}} C_{in} V_{dd}^2 \frac{f_o^2}{F_O \beta_r} \tag{4}$$

$$E_S(t_T >> \tau) = \frac{(0.5 - v_T)^{3/2}}{10 * 2^{3v_T + 2\alpha}} C_{in} V_{dd}^2 f_o \tag{5}$$

Here, $E_S(t_T << \tau)$ is the short circuit energy per switch of a gate with slow input, while $E_S(t_T << \tau)$ is the short circuit energy per switch of a gate with fast input, v_T is the normalized threshold voltage (V_{th}/V_{dd}), α is the velocity saturation index which is close to one at deep sub-micron technology, C_{in} is the input capacitance of the gate, V_{dd} is the supply voltage, f_o is the transistor driveability ratio of succeeding gates, and F_O is fanout (C_{in}/C_{out}).

The third term is the *static* power consumed due to *leakage* current through the transistors, which in reality function as "imperfect" switches. There are two distinct leakage mechanisms, and the magnitude of each leakage current is proportional to the width of the transistor and depends on the logical state of the device. The first type of leakage, *subthreshold leakage*, occurs when a transistor that is supposedly in the off state actually allows a small current to pass between its source and drain. We let I_{sub} denote the subthreshold through a unit-sized (off) transistor. The second type, *gate leakage*, is the current that leaks through the gate terminal. We determine the unit leakage current using MASTAR [50]. In McPAT, the unit $I_{g_{on}}$ includes both the gate leakage current flows through the channel to the gate and the gate leakage current flows through the source/drain and gate overlap.

In order to model the leakage current for a circuit block with many transistors, we need to consider which logical state each transistor is in, then sum up the leakage current components for each. If a circuit block is in some logical state s, we can express the *effective width* of all (off) transistors exhibiting subthreshold leakage in that state as $W_{sub}(s)$; similarly, we can express the effective widths for gate leakage for on and off transistors as $W_{g_{on}}(s)$ and $W_{g_{off}}(s)$. If $\Pr(s)$ is the probability that a circuit is in state s, we can express the total leakage current for a given block as the average

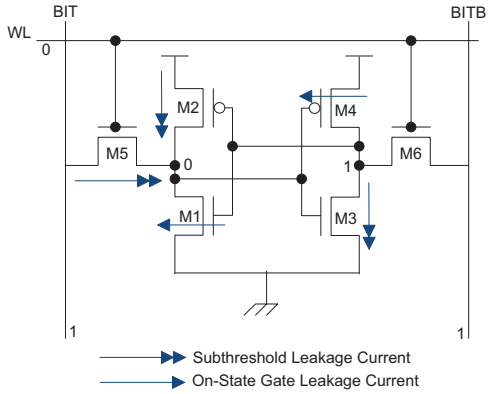


Figure 2: Leakage current paths in SRAM cell at idle state. Bitlines are precharged to VDD.

leakage current over all possible states S, as shown in Equation (6):

$$I_{leakage} = \sum_{s=1}^{S} \Pr(s)[W_{sub}(s)I_{sub} + W_{g_{on}}(s)I_{g_{on}} + W_{g_{off}}(s)I_{g_{off}}] \tag{6}$$

In McPAT, we calculate the leakage current for different states of each individual basic circuit block. McPAT assumes a circuit has equal possibilities to be in all states by default and computes total leakage. This probability distribution can also be overridden by the performance simulator when it has the ability to track the states of individual circuit blocks. Subthreshold leakage and gate leakage have a different leakage path even for the same circuit block in the same state. Gate leakage current when the device is on ($I_{g_{on}}$) differs greatly from the gate leakage current when the device is off ($I_{g_{off}}$). Since the off state gate leakage current is at least an order of magnitude less than the on state gate leakage current, McPAT ignores the off state gate leakage in the models. Figure 2 shows the subthreshold leakage and gate leakage paths in an SRAM cell at idle state with "0" being stored. Figure 3 shows subthreshold leakage and gate leakage paths in a NAND2 gate for all its possible states. When inputs are "11", subthreshold leakage reaches a peak since there are two transistors leaking in parallel. When inputs are "00", subthreshold leakage reaches a minimum because two leaking devices are stacked. Research in [39] shows that the stacking effects can reduce the subthreshold leakage significantly because of negative V_{gs}, a lowered signal rail V_{ds}, reduced drain induced barrier lowering (DIBL), and body effect. The stacking factor of a static CMOS gate with fan-in being equal to two as shown in Figure 3 can be obtained using Equation 7, where $\alpha = \frac{\lambda_d}{1+2*\lambda_d}$, λ_d is DIBL, S is subthreshold swing, and V_{dd} is the supply voltage. In the circuit level models, McPAT assumes maximum fan-in and fan-out do not exceed four to achieve good performance. The stacking factors of CMOS gates with fan-in of three or four are calculated using methods in [4].

$$StackingFactor = 10^{\frac{\lambda_d * V_{dd}}{s} * (\alpha - 1)} \tag{7}$$

3.2 Timing Modeling

Like the power model, McPAT's timing model breaks the system down into components and stages. While the power model requires only the capacitance to compute dynamic power, the timing model uses both resistance and capacitance to compute RC delays, using a methodology similar to CACTI [55] and the work by Palacharla et al. [43], with significant changes in implementation described later. McPAT determines the achievable clock frequency of a processor from the timing results of its components along the critical path. Every component has timing design constraints with two criteria:

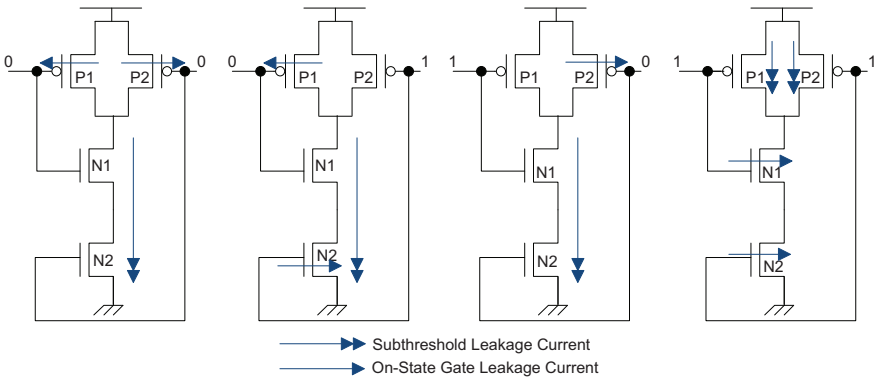


Figure 3: Leakage current paths in a NAND2 gate for all possible states.

throughput and latency. Throughput determines the cycle time of the component, while latency determines the access time. For components that can be pipelined, including caches and global interconnects, their achievable throughput sets the upper limit of the achievable clock rate of the processor. While for the components that cannot be pipelined such as the wakeup logic and result broadcast buses, their cycle time and access time are actually the same. Therefore, their latency and throughput set the upper limit of the achievable clock rate. When the latency or throughput of components along critical path cannot satisfy the target clock rate, McPAT will output warning messages and the user will need either change the configuration parameters or lower the target clock frequency.

3.3 Area Modeling

McPAT uses the analytical methodology described in CACTI to model area of basic logic gates and regular structures, including memory arrays (e.g., RAM, CAM (content addressable memory), and DFF (D flip-flop)), interconnects (e.g., router, link, and bus), and regular logic (e.g. decoder and dependency-checking unit). An algorithmic approach does not work well for complex structures that have custom layouts, such as ALUs. For these, currently McPAT takes an empirical modeling approach [17,44] which uses curve fitting to build a parameterizable numerical model for area from published information on existing processor designs, and then scales the area for different target technologies. McPAT computes the final die area of the target design by adding up the area of individual components. When adding area up, McPAT considers 10% placement and routing overhead. As mentioned in Section 2, there are three major places where interactions between components are considered. For these places, McPAT assumes the upper layer interconnect can be routed over lower layer components with minimum overhead.

3.4 Hierarchical Modeling Framework

McPAT's integrated power, area, and timing models are organized in a three-level hierarchy, as illustrated in Figure 4. McPAT's modeling framework provides comprehensive models for multicore/manycore processors from the architecture to the technology level. On the architectural level, a multicore processor is decomposed into major architectural components such as cores, NoCs (network-on-chips), caches, memory controllers, and clocking. On the circuit level, the architectural building blocks are mapped into four basic circuit structures: hierarchical wires, arrays, complex logic, and clocking networks. On the technology level, data from the ITRS roadmap [50] is used to calculate the physical parameters of devices and wires, such as unit resistance, capacitance, and current densities. As part of McPAT, we developed an enhanced CACTI that contains new CAM and fully-associative cache models, new technology models on

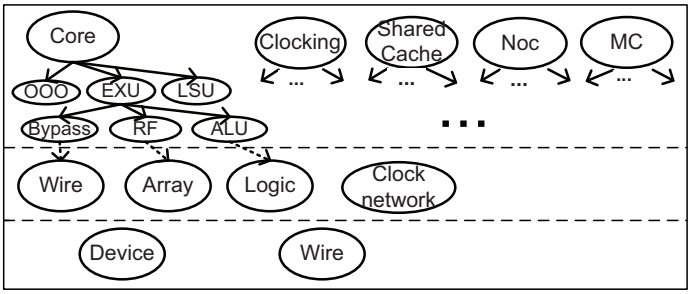


Figure 4: Modeling methodology of McPAT.

22nm double gate (DG) devices, new global interconnect models, new clocking models, new pipeline models and new short-circuit and leakage models. McPAT is tightly coupled with the enhanced CACTI and calls CACTI to obtain partial results at different modeling levels. In this technical report, we will focus on McPAT. All the details of the new features of the enhanced CACTI will be described in the CACTI 7.0 technical report [1] to be released.

4 Modeling Power-saving Techniques

McPAT models two major power saving techniques: clock gating to reduce dynamic power and power-saving states to reduce static power, which results in another distinguishing feature of McPAT—its ability to model advanced power management techniques, such as the P- and C-state [40] power management of modern processors. This allows the simulator to react to simulated power or thermal sensors (assuming a temperature model attached to the backend), by changing voltage and frequency settings, or by invoking one of multiple power-saving states on idle circuit blocks. Architects can use the framework to model power management alternatives.

4.1 P-state modeling

A P-state is an operational state, meaning that the core or processor can perform useful work in any P-state. It is a combination of clock frequencies and supply voltages. By reducing its clock frequency and/or supply voltage, the core or processor can achieve a much lower power profile while still being active although the processing speed will be lower. The modeling of P-states is based on the McPAT's ability to model clock gating and its flexible XML-interface. Since McPAT models dynamic energy per access per port for most components, it inherently models clock gating. Moreover, McPAT models the actual clock gating buffer circuitry at the distribution network heads. Combining these two features, McPAT can model clock gating schemes accurately.

After calling McPAT to finish initialization, a performance simulator can pass statistical information and invoke McPAT anytime during the simulation, and McPAT will calculate the corresponding power dissipation for the particular period and send it back to the performance simulator when required. When a performance simulator calls McPAT to compute the runtime power for a given period, the performance simulator can change Vdd and clock frequency settings. In this way, the performance simulator can apply P-state management techniques.

4.2 C-state modeling

A C-state is an idle state that is characterized by the amount of power consumed during the state and the latency and power consumption to enter and exit the state. As in modern processors [40], C0 is the state that no power-saving

S1S0	VG	Mode
00	0	Snore
01	V1	Dream
10	V2	Sleep
11	Vdd	Active

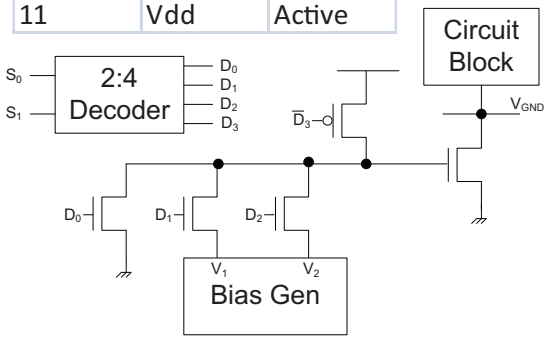


Figure 5: Circuit for multi-mode power-saving state control based on [2].

techniques are applied, where C1 to CN are different C-states with different static power saving levels. In C-states (except C0), the amount of leakage power can be reduced greatly. However, circuits in some deep power-saving modes cannot retain circuit states. For example, contents of a cache will be lost if the cache is in a deep power-saving mode. McPAT's ability to model C-states is supported by its ability to model power-saving states with multiple sleep modes as described in [2]. The user can specify power-saving modes for various components. A performance simulator can apply C-state management techniques to instruct the core or processor to enter a particular C-state during the simulation.

4.2.1 Power-saving State Modeling

In order to Model C-states, McPAT models a circuit used to support power-saving states inside each component and the wake-up overhead in terms of both timing and power. The circuit to support power-saving states is called a sleep transistor. The term sleep transistor describes either a PMOS or NMOS high Vth transistor that connects an external power supply to an internal circuit power supply which is commonly called virtual power supply. The sleep transistor is controlled by a power management unit to shift the voltage level of the virtual power supply so that the circuit can be put into different power-saving states when idle. A PMOS sleep transistor is used to switch the Vdd supply and hence is named as a header switch. An NMOS sleep transistor controls the Vss supply and hence is called a footer switch. A PMOS-based header switch can control a virtual Vdd supply, while an NMOS based footer switch can control a virtual ground. Circuit designers use footers, headers, or both to achieve optimal trade-offs between performance and overhead. PMOS transistors are less leaky than NMOS transistors of a same size. However, the advantage of a footer switch is its high drive and hence smaller area.

McPAT models the circuit used to support multiple power-saving states according to [2], where NMOS-based footer switches are used. In our modeling framework, there are four operating modes: active, sleep, dream, and snore. As shown in Figure 5, the circuitry uses footer devices and adjusts their gate biases to achieve different states. Having different sleep modes allows tradeoffs between the power saving and the wake-up overhead with respect to wakeup power and delay. For example, dream mode can save 50% more static power than sleep mode, but at the expense of twice the wake delay and three times the wake-up energy. As shown in Figure 5, the active mode is the state when the footer transistor is fully turned on. Because of the large width of the footer transistor, virtual ground V_{GND} has no significant difference from the real ground. The snore state is when the footer transistor is completely off with $V_G = 0$.

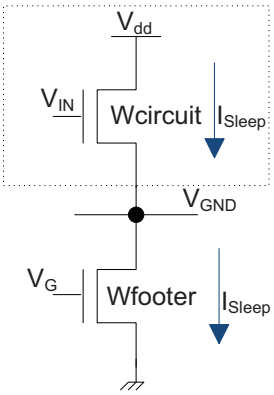


Figure 6: Footer sleep transistor model based on [2].

Figure 6 shows the model of a footer device associated with a circuit block. In order to match the effective width of the circuit block, the width of the footer transistor most usually is large. Therefore, the footer transistor dominates the wake-up penalty in terms of timing and power. The wake-up time and power when exiting a power-saving state are modeled using Equation 8 and Equation 9, where T_{wakeup} is the wake-up time, E_{wakeup} is the wake-up energy, V_{GND} is the virtual ground voltage level, C_{footer} is the capacitance of the footer transistor, $C_{circuit}$ is the total capacitance of the circuit on the charge/discharge path when entering/exiting power-saving states, and $I_{ON,F}$ is the saturation current of the footer device when it is on. The same delay and power consumption also occur when the circuit block enters the power-saving state. T_{wakeup} cannot be in parallel with any other operations since the circuit has to wait until its virtual ground has been completely discharged.

$$T_{wakeup} = \frac{(C_{circuit} + C_{footer}) * V_{GND}}{I_{ON,F}} \tag{8}$$

$$E_{wakeup} = \frac{1}{2}(C_{circuit} + C_{footer})V_{GND}^2 \tag{9}$$

$$V_{GND} = \frac{-V_G + S * log_{10}(\frac{W_{circuit}}{W_{footer}}) + Vth_F - Vth_C + \lambda_d * V_{DD}}{2 * \lambda_d} \tag{10}$$

$$\frac{I_{sleep}}{I_{active}} = 10^{-\frac{\lambda_d * V_{GND}}{S}} \tag{11}$$

$$P_{static} = P_{Cstatic} + P_{Fstatic} \tag{12}$$

$$P_{Cstatic} = (V_{DD} - V_{GND}) * I_{sleep} \tag{13}$$

$$P_{Fstatic} = V_{GND} * I_{sleep} \tag{14}$$

V_{GND} is calculated using Equation 10, where Vth_C and Vth_F represent the threshold voltages of the logic circuit and the footer device respectively, λ_d is DIBL, S is subthreshold swing, V_G is the gate voltage of the footer device, and V_{DD} is the supply voltage. Equation 10 shows that the steady state of V_{GND} linearly depends on footer gate voltage V_G, with

a negative slope. The upper limit of V_{GND} is obtained by setting $V_G = V_{DD}$ in Equation 10, which is about 80% of V_{DD}.

Among all the power-saving states, only the *sleep* mode is state retentive. The work in [2] chooses sleep and dream states to be the points that are evenly spaced in the wake-up penalty vs. leakage current graph. However, this choice cannot guarantee that the sleep state can retain the state of the circuit blocks since the circuit block's rail-to-rail voltage has to be larger than a certain value to preserve the data in standby. Therefore, we define the sleep state as when $V_{GND} = 10\% * V_{DD}$, which causes the drop of the circuit block's rail-to-rail voltage within 10%. Then, the *dream* mode is defined as a middle spot between *sleep* and *snore*. With all the V_{GND} calculated, the bias V1 and V2 can be obtained using Equation 10.

When the circuit block is in any of the power-saving states (sleep, dream, and snore), the footer transistor is turned off ($V_G < Vth_F$), operating in a weak inversion region. The leakage currents that follow through the circuit and the footer transistor are the same as shown in Figure 6. According to [2], the sleep leakage current to active leakage current can be expressed by Equation 11, where I_{sleep} is the leakage current in power-saving states, and I_{active} is the *leakage current* in active states. By changing V_{GND}, we can change the leakage current, and thus change power-saving states. When a circuit block is in a power-saving state, its static power can be computed using Equations 12 through 14, where P_{static} is the total static power, $P_{Cstatic}$ is the static power of the circuit block, and $P_{Fstatic}$ is the static power of the footer transistor. Our modeling results show that the sleep state can cut the static power of SRAM arrays in half while still retaining data in standby. This fits well with the published data of the 50% reduction of leakage power in the Intel 16MB L3 cache [10] because of using sleep transistors.

4.2.2 Models in Logic and Memory Structures

The circuit block shown in Figure 6 can be a single gate, a circuit block (e.g. an adder or a cache block), or a complete unit. When the sleep transistor is implemented in every single gate (cell), it is a fine-grain power gating style and called MTCMOS. On the other hand, when the sleep transistors are shared in a circuit block, it is a coarse-grain power gating style. The advantage of the fine-grain sleep transistor implementations is that the virtual power nets are short and hidden in the cell. However, the fine-grain sleep transistor implementation adds a sleep transistor to every MTCMOS cell, which results in a significant area increase. The fine-grain sleep transistor implementation also suffers from high IR-drop variation in the cell and hence performance variation. The main advantage of the coarse-grain power gating is that sleep transistors share charge/discharge current. Consequently, it introduces less IR-drop variations than the fine-grain implementations. Also, the area overhead is significantly smaller due to charge sharing among the sleep transistors. Most power-gating designs prefer the coarse-grain sleep transistor implementation than the fine-grain implementation which incurs large area penalty and higher process-voltage-temperature (PVT) sensitivity. Therefore, we use coarse-grain sleep transistor implementation in McPAT's models.

As shown in Equations 8 to 14, sizing of the footer transistor is critical to achieve good trade-offs between static power saving and wake-up penalty. According to [2], we choose the width of footer transistor (W_{footer}) as 15% of the accumulated width of transistors that in the charge/discharge path in the circuit block when exiting the power-saving states. The charge/discharge path depends on the inputs of the circuit block, and we assume the pull-down network is open when the circuit clock wakes up. Therefore, $Wfooter = 15\% * \sum C_{NMOS}$. As $Wfooter$ is usually large, it can be implemented in a group of parallel connected transistors. The wake-up time and energy remain the same. Although area also remains approximated the same, but the area efficiency can be better since narrower and longer channels can be used for virtual ground and the aspect ratio of the virtual ground channel can be adjusted according to the width of the circuit block. As shown in Figure 7, the number and distribution of sleep transistors depends on the area of the function unit/circuit block. McPAT tries to fill the width of a circuit first then increase the height to accommodate sleep

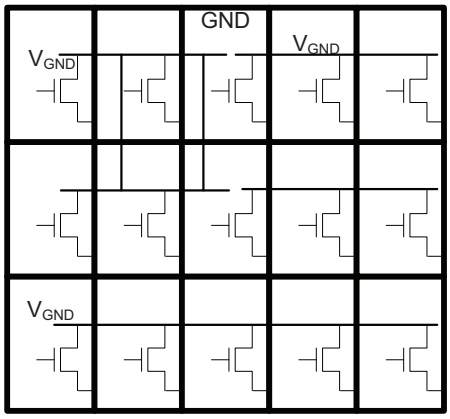


Figure 7: Distributed sleep transistor.

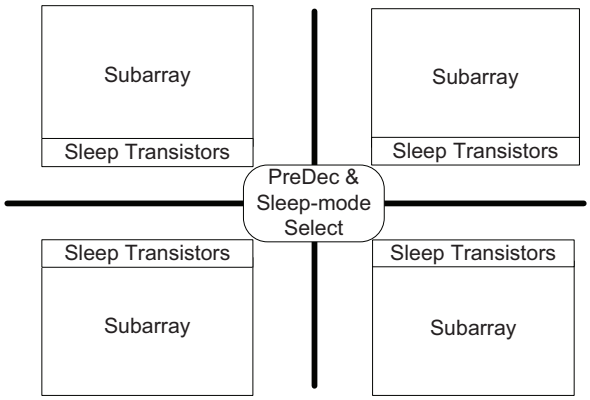


Figure 8: A mat with four subarrays with associated power-saving logic.

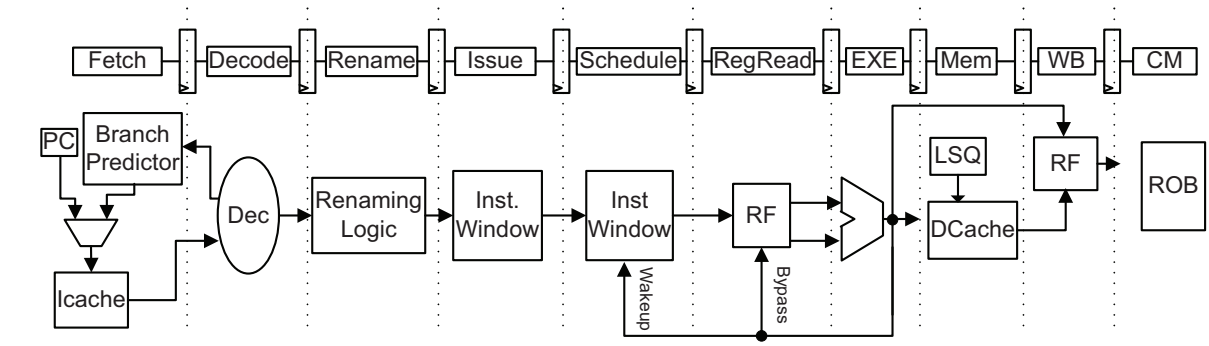
transistors.

In a memory array, the footer transistor can be implemented on a per row basis [37]. But this implementation tends to have a larger IR drop or more area overhead. Our models follow a section-based implementation as in [63], where the footer transistors are implemented inside each subarray as shown in Figure 8. Similar to the original memory models in CACTI, the pre-decoder block also has the 2-4 decoder to choose the sleep modes, and each memory subarray contains sleep transistors which result in area overhead, wake-up delay and wake-up power.

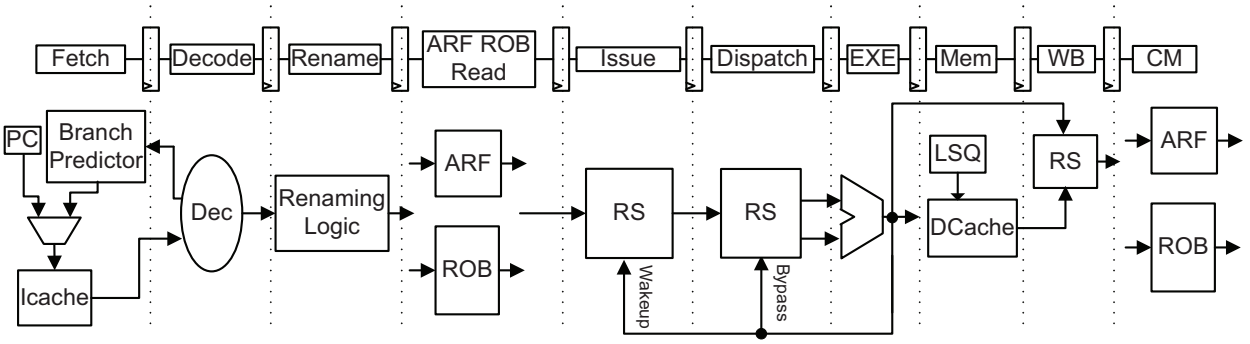
As in real industrial high performance processors, McPAT assumes that circuit blocks can retain states only when they are in *sleep* mode. If a user or performance simulator chooses to set hardware units into dream or snore state, McPAT assumes that all the states/contents in the circuit blocks that are in *dream* or *snore* will be lost (McPAT does not model the extra hardware to save the circuit state since the state can be huge, e.g. contents of last-level cache). The McPAT user needs to make sure that the performance simulator is configured properly to reflect this wake-up overhead, for example the whole cache contents are flushed and the cold-start effects on the cache need to be appropriately reflected in the statistics that are sent from performance simulator to McPAT.

5 Architecture Level Modeling

This section introduces the architecture level models in McPAT. The highest level is the architecture level that represents the basic building blocks of a multicore processor system. McPAT is designed to be flexible enough so that it can



(a) McPAT model for physical register based OOO core.



(b) McPAT model for reservation station based OOO core.

Figure 9: Out-of-order core model in McPAT

work with various performance simulators, and it is also designed to be accurate enough so that the results are widely applicable. One big challenge of achieving these two goals simultaneously is that different performance simulators differ greatly on the level of detail of the statistics they can provide and therefore McPAT might not have a way to fill in all missing statistics necessary to compute runtime power. For example, SimpleScalar [6] only provides the number of memory instructions, while M5 [7] can give the number of accesses to a load queue and a store queue. In order to solve this problem, we made detail generic assumptions on the architecture level based on real designs of high performance processors.

5.1 Core

Figure 9 shows the McPAT's default out-of-order core models. The reservation-station-based OOO model is extracted from the Intel P6 [22] architecture, and the physical-register-file based OOO model is extracted from the Intel Netburst [19] architecture and the Alpha 21264 processor. By default, both OOO models are assumed to have ten pipeline stages in McPAT's modeling framework. In-order processor is modeled in a similar way but with only five stages including fetching, decoding, execution, memory access, and write back. A McPAT user can override the number of pipelines of both in-order and OOO processors. A core can be divided into several main units: an instruction fetch unit (IFU), an execution unit (EXU), a load and store unit (LSU), a memory management unit (MMU), a renaming unit and an out-of-order (OOO) issue/dispatch unit for OOO processors. Each of them can be further divided into hardware structures. McPAT models each units in detail at architecture level.

5.1.1 Instruction Fetch Unit (IFU)

McPAT models the IFU as three major parts: the instruction cache together with its cache controller, the instruction buffer, and the branch predictor. The instruction cache is mapped to memory arrays and computed using its internal CACTI module. The cache controller is modeled as three major structures: the miss buffer (miss status handling registers (MSHRs)) [30], the fill buffer, and the prefetch buffer. Almost all current high-performance processors support non-blocking caches. A plain cache can only handle one outstanding requests at a time. If a request is made to the cache and there is a miss, the cache pipeline must wait for the memory to supply the value that was needed, and until then it is "blocked". By adding a special miss buffer, the cache can store the missed memory requests into the miss buffer and continue working on later requests while waiting for memory to supply previous misses. Although usually the miss buffer is designed for load misses since load misses are much worse than store misses, McPAT assumes that both load and store misses will setup an entry in the miss buffer based on the miss address file (MAF) design of Alpha 21264 and the non-blocking cache of Intel P6 [22]. The miss buffer is modeled as a fully associative cache which has a non-tag portion and a CAM-based tag portion. The non-tag portion holds the data, the internal cache address, and the destination register, while the tag portion contains the address. McPAT assumes that each cacheable miss uses one CAM operation.

The fill buffer is also modeled as a fully associative structure using the internal CACTI module based on Sun Niagara [54]. The data portion contains cacheline-size entries to store data from the lower level memory (such as L2 or DRAM) before it fills the current cache. Addresses are stored in the CAM portion for maintaining the age ordering in order to satisfy coherency conditions. Although it contains a cacheline per entry, the block size of the fill buffer is not the instruction cache line size but only the size of the machine word. This is because the data bypass that allows the instructions pending on cache misses can enter the pipeline as soon as the critical machine-word arrives at the fill buffer. After all words arrive, the fill buffer assembles them into a cache line and then write the cache line into the cache. As most of the simulators do not model miss buffers and fill buffer, McPAT uses inherent internal relationship between the buffers and cache accesses patterns provided by the performance simulator to model the operations on miss and fill buffers. Every cache miss will access the miss buffer. Every data written into cache will access the fill buffer. When data comes back to the fill buffer, this means a miss has been serviced, and the miss buffer is accessed again to read out the instruction that can be re-issued.

Prefetch logic is also an important part of cache controller hardware and modeled in McPAT. Prefetch instructions and/or data before they are requested by the processor can hide the memory latency greatly. Since directly prefetching data into the caches may cause cache pollution as the prefetched data may replace the cache entry resulting in later cache misses, McPAT assumes that an external buffer is used to hold the prefetched data as in Sun Niagara design [54]. Unless the performance simulator provides the prefetch information, McPAT assumes that the processor always fetches two blocks on a miss: the requested block and the next consecutive block as in Sun Niagara [54]. The requested block is placed in the instruction cache when it returns, and the prefetched block is placed in the instruction prefetch buffer. All the modeled buffers have the same number of ports as the plain instruction cache which has one port by default. McPAT does not model the state machine inside the cache controller in the current version.

Branch Predictor hardware is modeled as three main parts: the branch predictor, the return address stack (RAS), and the branch target buffer (BTB). The default predictor in McPAT is modeled as the tournament predictor as in Alpha 21264 [26]. The predictor has three RAM structures: the global predictor, the local predictor, and the chooser. Both the global predictor and the chooser are single-banked RAM arrays, while the local predictor is a double-banked array, with one bank being used for first-level prediction and another bank being used for second-level prediction. Branch target buffer is modeled as a normal cache. The RAS that is accessed when function calls and returns happen is also modeled

as a RAM array. The number of ports of all array structures in the branch prediction hardware is assumed to be the same as that of the instruction cache.

An instruction buffer is a part of IFU to temporarily store instruction stream before sending it into decoding stage, and it is mapped to a RAM array at the circuit level.

5.1.2 Renaming Unit

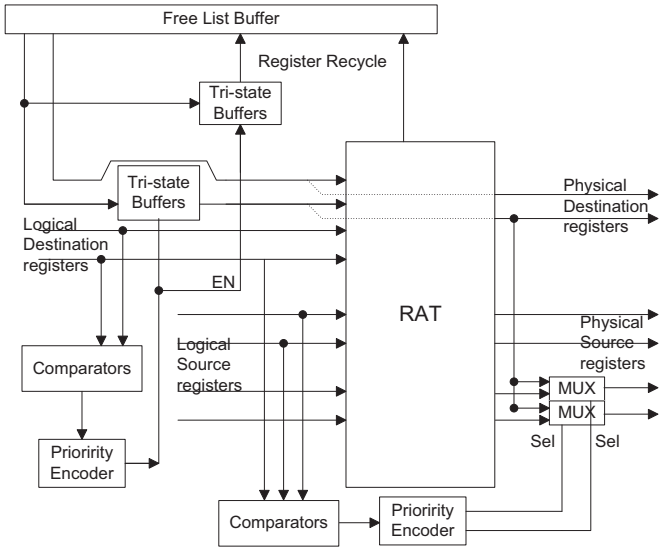


Figure 10: Architectural models of renaming logic in McPAT.

The renaming units are very important for OOO processors. They consumes significant portions of power and have high power density [16]. Figure 10 shows the renaming logic model in McPAT. As shown in Figure 10, there are three major parts in a renaming logic: the register alias table (RAT), the dependence check logic, and the free list buffer. The free list buffer is modeled as a multiported FIFO mapped to the SRAM array at the circuit level. The RAT is modeled as RAM or CAM array depending on the renaming scheme and is mapped to RAM or CAM structure at the circuit level. Dependency check logic is modeled analytically as random logic. McPAT also considers the instruction decoder as part of this unit since its outputs are steered into the renaming logic and later pipeline stages.

The renaming operations proceed as follows. At every cycle a batch of instructions are fetched and decoded. These instructions form the current renaming pool. The rename logic is responsible for removing the false data dependencies including write-after-write (WAW) and write-after-read (WAR) and maintaining true data dependency of read-after-write (RAW) among instructions in the current renaming pool and in-flight. During the renaming process, the architectural destination register of each instruction in the current rename pool is renamed with a physical register, taken from a pool of available registers (free list), so that no two instructions with the same destination register enter the processor pipeline. When physical registers are assigned to the architectural destination registers, the designators of all the architectural destination registers of instructions in the current renaming pool need to be compared against each other to avoid mapping same physical register to different architectural registers. The comparison is done by the dependency check logic as shown in left side of Figure 10. As a result, the WAW data dependencies are removed. Since each logical destination register is assigned a new physical register from the free list, this physical destination register is different from the source registers of instructions in-flight and instructions in the current renaming pool. Therefore, WAR data dependencies are also removed, and only RAW dependencies are left. The RAT is maintained to translate architectural source registers

of an instruction into physical register designators. In every clock cycle, the name mapping of the instruction source registers is read from the RAT (by using the architectural register designator as the index in RAM-based RAT or by associative search using the designator as the key word in CAM-base RAT), while the new mapping of the destination registers is written in the RAT. At the same time, the overwritten designators must be read and inserted in the free list buffer when the last in-flight instruction that uses them commits.

In order to maintain RAW dependencies, another section of the dependence check logic is used to detect the cases where the registers being mapped are destination registers of an earlier instruction in the same rename pool. The dependency check logic compares source architectural register designators of instructions in the same rename pool against each other. A match in the dependency check logic indicates that an RAW is within the current renaming pool. The output MUXes of the dependency check logic select the mapping used for destination register in thecurrent renaming pool, instead of the physical register found in the RAT. At the same time, the new designator of the source register overrides the old physical register mapping in RAT. In case of multiple matches in the dependence logic, a priority encoding logic selects the latest match in fetch order as the valid mapping of the source register. This source physical register designator is sent to later pipeline stages. If dependency check logic finds no matches and there is a valid mapping found in the RAT, the mapping found in RAT is passed to later pipeline stages and the RAT entry stays valid.

5.1.2.1 Register Alias Table (RAT)

McPAT supports both RAM-based RAT as in MIPS R10K [61], Intel P6 architecture [22], and Intel NetBurst architecture [19] as well as CAM-based RAT as in Alpha processors [26]. The RAM-based RAT is modeled as an SRAM array with the number of entries being equal to the number of architectural registers. The entries are indexed by the designator of the architectural registers. The physical register designators for the corresponding renamed architectural registers are stored in the entries. For the CAM scheme, the RAT is modeled and mapped to CAM array models at circuit level with the number of entries being equal to the number of physical registers (or reorder buffer entries). Each entry has two data fields: one data field contains the designator of the architectural register that is mapped to this physical register, and the other data field contains a valid bit to indicate that the current mapping is valid. During the renaming operation, the designator of the architectural register is applied on the search line of the CAM. An associative search is performed to find a match between the data on search lines and the data stored in the cross-coupled inverters inside the CAM cells. If there is a match and the valid bit is set, the match line will be activated. An external encoder will encode the match line into the designator of the corresponding physical register and send the designator of the physical register to components that need the information. The CAM scheme is believed to be less scalable than the RAM scheme since the number of entries of CAM-based RAT grows proportional to issue width [43]. However, the CAM scheme is advantageous in terms of checkpointing, and this advantage can be seen from McPAT results.

For physical register-based cores, the physical register file holds both speculative and non-speculative data. However, the non-speculative subset of the data must be explicitly denoted when the the processor state must be saved in case of interrupts/context switches. This requirement of denoting the subset of the non-speculative data leads to the dual-RAT technique [19]. McPAT models the dual-RAT scheme in its physical-register-based core models. A front-end RAT (FRAT) is used for register renaming as usual, while a retire RAT (RRAT) is used to store the current mapping for architectural registers. Since the FRAT contains the latest renaming information, there is no change for operation of the renaming unit with single RAT. An RRAT is only accessed during the instruction commit stage and does not affect the FRAT. The FRAT is modeled using both RAM and CAM schemes in McPAT. The RRAT is always modeled as RAM

structure with its number of write ports equals to the commit width of the modeled core.

McPAT also models checkpointing hardware in renaming logic based on the designs of global checkpointed RATs in [15, 46]. The renaming unit needs to be recovered to correct non-speculative states when branch mispredictions happen. There are three major techniques used in current higher performance OOO processors for renaming unit checkpointing and branch misprediction recovery: (1) using a reorder buffer (ROB) as in the MIPS R10K to save non-speculative mapping, (2) using global checkpoints (GCs) to take a complete snapshot of the RAT upon renaming a branch instruction as in the Alpha 21264, and (3) not to checkpoint RAT to achieve a simpler designs at the expense of some sacrificing performance as in the Intel P6 and Netburst architectures.

In the ROB checkpointing scheme, the recovery time is proportional to the number of instructions on the wrong path. This approach also increases the size and power of the reorder buffer. McPAT currently does not support this approach but support the other two checkpointing schemes. The major difference between the use of checkpointing or not is that processors without RAT checkpointing must stall the renaming unit and wait for the OOO core drains out none-speculative instructions by letting these instructions commit to architectural register file or RRAT. A processor with RAT checkpointing can start to restore renaming map immediately after the misprediction is detected. A processor without renaming checkpointing have to pay the penalty of execution-to-retirement delay, which is a variable factor that depends on the number of instructions between the executing branch and the retirement point and can be huge when the machine is full of instructions. McPAT supports both approaches to enable the comparison between alternative approaches.

Since most performance simulators do not model the internal operations of renaming units in detail, McPAT models the internal operation by itself and only requires the access counts to the whole renaming unit. McPAT assumes the RAT needs $2 * W$ read ports [53] and W write ports, where W is the issue width.

5.1.2.2 Dependency Check Logic (DCL)

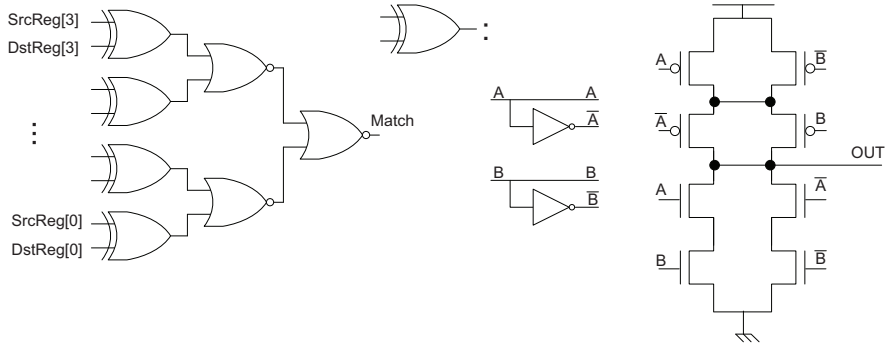


Figure 11: Comparator Sets of dependency check logic (DCL) modeled in McPAT

As shown in Figure 10, there are two sections of dependency check logic. The first section checks the WAW dependency among destination registers of instructions in current renaming pool. This dependency check logic (DCL) has three major parts: the comparators, the priority encoders, and the tri-state buffers. When dependency is detected among destination registers, the output of the priority encoders act as the enable signals, disabling the write path to the RAT and enabling the recycle path to the free list buffer. The comparators are modeled based on [8]. We model a a scalable priority encoder as shown in Figure 12. Since McPAT supports processors with issue width up to eight it is important to have the DCL logic scalable so that it will not limit the clock rate. In the priority encoder, the signal with

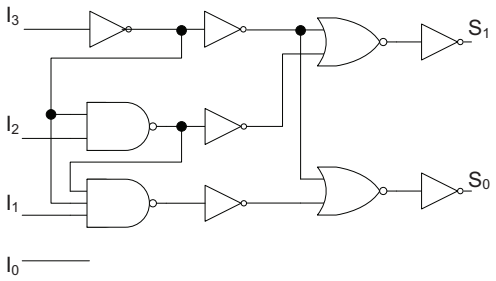


Figure 12: Priority Encoder of DCL modeled in McPAT

lowest priority, I_0 does not connect to the main logic. When I_0 is set, the output of the priority encoder will still be 00, but the match signal from comparator sets will indicate that data dependencies are detected on instruction associated with I_0. The circuit of comparator sets is shown in Figure 11. The total number of comparator sets of this section is shown in Equation 15, where S is the total number of sets needed for destination register renaming and W is the decode width.

$$S = W * (W - 1) \tag{15}$$

The second sections of DCL is shown at the bottom of Figure 10. This section of dependency check logic is modeled as multiple sets of comparators, priority encoders, and multiplexors according to [8]. It detects RAW, chooses the correct mapping of the source registers, and sends them to later pipeline stages as mentioned earlier. The number of the comparator sets used for second stage of the dependency check logic is twice the number used in comparing the destination registers. Therefore, the total number of comparator sets is shown in Equation 16, where S is the total number of sets needed for destination register renaming and W is the decode width.

$$S = 3 * W * (W - 1) \tag{16}$$

5.1.2.3 Power, Area, and Timing of Renaming Logic

When modeling timing, the internal CACTI module is used to model the array structures, namely, the RAT and the free list buffer. Since McPAT models power and area of target processors for given timing constraints, it focuses on the critical path for timing.

The total delay of the renaming logic is shown in Equation 17, where $T_{FreeList}$ is the access time of the free list buffer, $T_{DCLsec1}$ is the delay of the first section of dependency check logic that is used to detect WAW, $T_{DCLsec2}$ is the delay of the second section of dependency check logic that is used to detect RAW, $T_{RATread}$/$T_{RATwrite}$ is the read/write latency of the RAT (in dual RAT scheme, FRAT determines the timing on critical path), and $T_{DCLSec2MUX}$ is the latency of the MUXes at the final stage of the dependency logic. The total delay must be less than the target clock period in order to satisfy the timing constraint.

For each instruction that has register source operands, McPAT assumes it accesses dependency check logic and reads the RAT once for each of its source operand registers. For each instruction that has register destination operands, McPAT assumes it accesses dependency check logic and writes the RAT once for each of its destination operand registers. In a dual RAT scheme, RRAT is updated when an instruction commits. Energy consumed by each access is computed as shown in Equation 18, where E_{DCL} is the energy per access of the dependency check logic, E_{RAT} is the energy per access for the RAT which depends on the operation type (in a dual RAT scheme, E_{RAT} is the total energy of FRAT and RRAT

for the same instruction), and $E_{FreeList}$ is the energy per access of the free list buffer.

The area of the renaming unit is shown in Equation 19, where A_{DCL} is the total area of the two sections of the dependency check logic, A_{RAT} is the area of the RAT (in dual RAT scheme, A_{RAT} is the total area of FRAT and RRAT), and $E_{FreeList}$ is the area of the free list buffer. We consider 10% overhead for macro level placement and routing when putting all individual components together to form the renaming logic.

$$T = max(T_{DCLsec1}, T_{FreeList}) + max(T_{RATread}, T_{RATwrite}), T_{DCLsec2} + T_{DCLSec2MUX} \tag{17}$$

$$Energy_{RenamingLogic} = E_{DCL} + E_{RAT} + E_{FreeList} \tag{18}$$

$$Area_{RenamingLogic} = (A_{DCL} + A_{RAT} + A_{FreeList}) * 1.1 \tag{19}$$

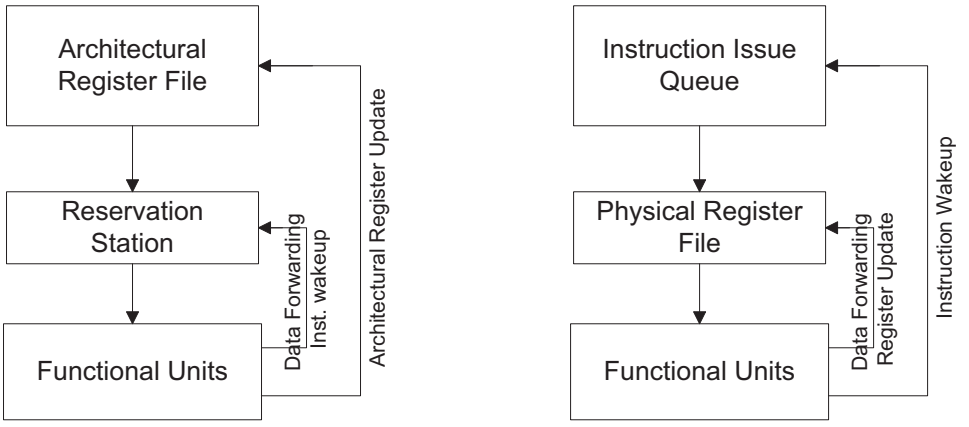
5.1.3 Scheduling Unit and Execution Unit

McPAT supports detailed and realistic models of the scheduling unit that are based on existing high-performance OOO processors. McPAT models both reservation-station-based (data-capture scheduler) architectures such as the Intel P6 [22] and physical-register-file-based (non-data-capture scheduler) architectures such as the Intel Netburst [19] and the DEC Alpha [26]. For an in-order processor, the scheduling unit degenerates to a simple instruction queue. In McPAT, the scheduling unit has three major components: the scheduling window, the reorder buffer, and the broadcast buses.

Figure 13 shows the conceptual view of both the reservation-station-based (data-capture scheduler) cores and physical-register-file-based (non-data-capture scheduler) cores. The two core models have same in-order frontends that include instruction fetch stage, decoding stage, and register renaming stage. After passing the frontend, instruction streams enter the OOO scheduler where the key difference of the two models resides. Renamed instructions are issued into the scheduling window, waiting for their source operand and functional units to be available before they can be dispatched to the execution stage. In the reservation-station-based model, the reorder buffer holds the speculative register data and the architectural register file holds the non-speculative data. Available source registers are read out before instructions are issued into the scheduling window. The designators of the unavailable source registers are also copied into the scheduling window and are used for matching the results from functional units and waking up appropriate instructions.

5.1.3.1 Reservation Station

The reservation station is modeled as a fully associative structure with a CAM array and an SRAM array. The CAM array holds the designators of the source register operands and is multiported with $2 * W * Nsource$ search ports and W write ports, where W is the issue width, $Nsource$ is the number of source register operands per instruction. The SRAM array holds the information for instruction execution, including the instruction opcode and the source operands' data. Write accesses happen when instructions are issued to reservation stations, and search operations happen when results are available from the broadcast bus. Instructions will be woken up once the comparisons for source operand designators succeed. The CAM search operation serves as the wakeup logic for the reservation station. Since it is possible that multiple instructions become ready in the same cycle, selection logic is modeled according to [43]. In order to have dependent instructions issued consecutively, wakeup and selection must be done within one cycle. The worst case latency of the reservation-station-based design is shown in Equation 20, where $T_{CAMsearch}$ is the latency of search operation in the CAM, $T_{RSwrite}$ is the write latency of the whole reservation station array (both CAM and RAM),



(a) McPAT models of Reservation Station based OOO Scheduler

(b) McPAT models of Physical Register File based OOO Scheduler

Figure 13: Conceptual view of architectural models in McPAT of reservation-station-based and physical-register-file-based OOO scheduler [51]

and $T_{InstSelection}$ is the latency of the selection logic. This latency must be less than the clock cycle time, and McPAT tries to optimize the reservation station to achieve this goal. A reservation station requires a large scheduling window in order to hold both tags and data in the same array structure. This is not easy for McPAT to satisfy the timing constraints of both the latency and throughput.

$$T = max((T_{CAMsearch} + T_{InstSelection} + T_{RAMread}), T_{RSwrite}) \tag{20}$$

5.1.3.2 Instruction Issue Window

In the physical-register-file-based model, the physical register file holds both the speculative and non-speculative data. Only designators of the source operands are stored together with the instructions in the scheduling window. The tags are used to wake up instructions, and an instruction will read the register file to obtain data right before it is sent into the function units. The instruction issue queue in the physical-register-file-based model is modeled similarly to the reservation station in the reservation-station-based based model. The major difference is that the data portion of the instruction issue queue does not have the values of registers, which results in a smaller data array.

5.1.3.3 Result Broadcast Bus

As shown in Figure 13, reservation-station-based models combine result forwarding and instruction wake up into a single broadcast bus. In McPAT, the broadcast logic is modeled based on [43], including repeated wires, MUXes, and tri-state buffers. Also according to the die photo of AMD Opteron processor (Hammer), we assume that broadcast buses route across all function units, the reservation station, the architectural register files, and the ROB. The buses are modeled using hierarchical wires, as described in Section 6. The total latency of the results bus must be less than that of the clock period.

As shown in the Figure 13, the physical-register-file-based model separates the wakeup path of the control flow from the forwarding path of the data flow. Therefore, McPAT models two different portions of the result broadcast bus. The data portion of the broadcast buses route over the functional units and physical register files, while only the designator portion of the broadcast bus needs to route over the instruction issue queue and ROB in addition to the functional units

and physical register files.

5.1.3.4 Reorder Buffer (ROB)

The reorder buffer (ROB) is modeled as a circular RAM array in McPAT with the number of entries being equal to the number of in-flight instructions. McPAT assumes each issued instruction (including the load/store instructions) has been assigned an entry in the ROB. Each entry of ROB contains the designator, data from the destination register, and the instruction address. In McPAT's models, the reservation-station-based-model ROB in has $3*W$ read ports and W write ports, assuming issue width is equal to commit width. The physical-register-file-based-model ROB is modeled in a similar way except the ROB does not contain the register values. The smaller physical-register-file-based scheduling window has relaxed the timing constraints compared to the reservation-station-based models, which in turn enables a higher achievable processor/core clock rate. The latency of ROB must be less than the clock period set by the user.

5.1.4 Execution Unit

The execution unit and the scheduling unit are tightly coupled. The execution unit includes functional units and register files. It also shares the broadcast buses with the scheduling unit. The execution unit contains ALUs, FPUs, and register files. In our hierarchical framework, the ALU and FPU are mapped to the complex logic model at the circuit level.

5.1.4.1 Register Files

Register files are mapped to the array model in McPAT. The architectural register file and physical register file have the same number of ports, $2*W$ read ports and W write ports, where W is the issue width. The access time of register file must be less than the that of clock period of the core, and the throughput must be at least equal to the clock frequency of the core clock rate. McPAT also models windowed register files. Windowed register files are not alternatives of register renaming, rather, they are techniques to improve the performance of a particularly common operation, the procedure call. These techniques can usually be found in Sun processor families. Unfortunately, windowed register files make register renaming extremely difficult to implement efficiently [52]. Without register renaming, there is little to be gained from OOO execution. Therefore, McPAT assumes that once the windowed register file is selected by user, the renaming unit and OOO logic will be disabled. The windowed register file is modeled as a large RAM array with single read and write port. The windowed register file is accessed on every function call and return with multiple reads and writes, assuming 24 out of 32 registers will be stored or restored into the working architectural register files.

Since McPAT allows users to focus on high level architecture without worrying about low level configurations, it also provides default values for the physical register file size. The total number of physical registers in the worse case is shown in Equation 21. However, this worst case is too pessimistic. By default, McPAT assumes the total number of physical registers as shown in Equation 22. According to [57], this assumption works well for processors with a reasonable numbers of threads.

$$Num_{PhysicalRegisters} = Num_{In-flightInstructions} * 3 \tag{21}$$

$$Num_{PhysicalRegisters} = Num_{threads} * Num_{ArchiRegs} + 100 \tag{22}$$

5.1.5 Memory Management Unit

The memory management unit contains two main structures: a instruction TLB and a data TLB. Currently McPAT supports single level TLB structures. TLBs are mapped to array structures at circuit level and modeled as fully associative caches. We assume that all L1 caches are virtually indexed and physically tagged. Therefore, the CAM portion of TLBs holds the virtual address, and the data portion holds the tag of the physical address for the cache. TLBs have the same number of ports as their corresponding caches. The timing requirement on a TLB is to finish an access within a single cycle.

5.1.6 Load and Store Unit

McPAT models the load and store unit (LSU) as three components: a data cache, a load queue, and a store queue. A data cache is modeled in the same way as the instruction cache in IFU except that a write back buffer (WBB) is modeled as part of the cache controller if the data cache is set to be write-back. The WBB is used to store the evicted dirty cache lines. The evicted lines are streamed out to the lower level memory opportunistically. The WBB is divided into a RAM portion, which stores the evicted data until it can be written out, and a CAM portion, which contains the address.

Load queues and store queues are designed assist memory renaming and permit the out-of-order execution of memory instructions. Based on modern architectures such as the Intel P6 and Netburst, the load/store queue is modeled as two separate queues in McPAT and has three functions: (1) The load and store queues buffer and maintain all in-flight memory instructions in program order. (2) The load and store queues support associative searches to honor memory dependences. A load searches the store queue to obtain the most recent store value, and a store searches the load queue to find any premature loads (store-load order violation). (3) In multicore processors, the load and store queues support associative searches to enforce memory consistency.

McPAT assumes that for all memory instructions the ROB still holds all load and store instructions in-order, in other words, McPAT does not support the combined ROB and load-and-store-queues approach as in MIPS 10K. Similar to the Intel P6 and Netburst architecture, McPAT also assumes that the instruction window (reservation station or instruction issue queue) will hold the load and store instructions until the their operands are ready (both data and address for stores) and then dispatch the them into LSU.

McPAT models two different types of load and store queues for in-order and out-of-order issued memory instructions. It is very important to model the two different types as this allows McPAT to reason about the statistics of load and store queues when the performance simulator cannot provide detailed information. McPAT models in-order issuing of memory instructions according to Intel P6 architecture. When a load enters the load queue to see if there are stores with conflicting addresses, all previous stores in program order are already in the store queue. Issuing loads and stores in-order ensures that all preceding stores to a load will be in the store buffer when load executes. This, however, limits the out-of-order execution of loads. Although a load instruction can be ready to be issued, a preceding store can hold up the issue of the load even when the two memory addresses do not alias. McPAT also models OOO issuing of memory instructions based on the Alpha 21264 processor. If loads and stores issued out-of-order, it is possible for some of the stores that precedes a load to still be in the scheduling window while the load has already enters the LSU, thus the load queue. Therefore, just searching the store buffer is not adequate for checking potential aliasing between the load and all its preceding stores. Therefore, loads are allowed to issue out-of-order and be executed speculatively, and all speculatively finished loads are stored in the load buffer. When a store issues, it performs an associatively search in the load queue using its address. A match in the load queue indicates that an older store is issued after a younger load to the same memory address. Thus, the load queue must squash the load since the load got the wrong data. All the subsequent

instructions stored in ROB after the load also need to be flushed.

Since McPAT is designed to model multicore processors, we assume that the load-load ordering is enforced at the hardware level as in the Alpha 21264 and MIPS R10K although this may also be handled by software. When a load executes, it searches the load queue to compare its address to the addresses of all loads. If there is a match with a younger load issued out-of-order, then the out-of-order-issued load and subsequent instructions are squashed and fetched again.

The sizes of modeled load and store queues in McPAT are determined by the maximum number of in-flight load and store instructions. The numbers of ports of the load queue and the store queue are determined by the number of memory instructions that can be issued per cycle. Specifically, the store buffer has: $2*L$ read(compare) ports and L write ports, where L is the number of memory instructions can be issued per cycle. The load buffer has $2*L$ read(compare) ports and L write ports. In order to maintain load-store and load-load order, McPAT considers that for each memory instruction there are multiple inherent accesses to the load and store queue as shown in Table 1

	Load instruction	**Store instruction**
In-order issue	1 write op (send the load in the Q); 1 read op (update the ROB or bypass); 1 CAM op in store buffer; 1 CAM op in load buffer to maintain load-load op;	1 write op (send in the Q); 1 read op (update main memory);
Out-of-order issue	Same as above	Except above ops; 1 CAM op in load queue to find younger loads that must be squashed

Table 1: Activity factor assumptions on load and store queue in McPAT

5.1.7 Pipeline Logic

Pipeline logic is modeled as a row of registers at each pipeline stage. By default, we assume that 5 and 10 stages per pipeline are used for in-order and OOO processors, respectively as shown in Figure 9. User are also allowed to increase the number of pipeline stages. We assume that the main reason for increasing the number of pipeline stages is that the user defined target clock frequency cannot be satisfied during optimization. Therefore, we assume increasing number of pipeline stages for a certain function by X times will increase the number of inter-stage registers for the function by X times. The inter-stage registers are modeled based on implementations using NAND2 gates(positive-edge trigger) and using NOR2 gates (negative-edge trigger). In front of each group of pipeline registers, a NAND gate is modeled for clock gating.

5.1.8 Undifferentiated core

Other than the modeled units so far, there are other glue/control logic units on silicon, such as the trap logic that handles exceptions. Those logic units are hard to model since they are highly customized depending on the processor architecture and implementation. They may not be as active as other components such as reaming logic but they occupy die area and consume leakage power. They are modeled by the undifferentiated core function in McPAT.

An undifferentiated core is a core that does not have the components that are modeled analytically as mentioned above. The area of a native core is obtained by measuring die photo of Sun processors including the 90nm Niagara [34] and the 65nm Niagara 2 [41] as well as Intel processors including the 90nm Pentium 4 [48], the 65nm Xeon Tulsa [45], the 65nm Merom [47] and the 45nm Penryn [14], and then curve-fitting to get equations with respect to processor type (in-order vs. OOO), number of pipeline stages and issue width. The transistor density of the undifferentiated core is obtained from the data of Intel Penryn processor [14], which is used to calculate leakage power for the undifferentiated core.

5.1.9 Models of Multithreaded Processors

McPAT also models the power, area, and timing of multithreaded processors whether in-order (e.g., Sun Niagara) or out-of-order (e.g., Intel Nehalem [31]). There are three different types of multithreading: coarse-grained multithreading (CGMT), fine-grained multithreading (FGMT), and simultaneous multithreading (SMT). A CGMT processor issues instructions from one thread until the thread is blocked When a thread is blocked, a CGMT processor switches to another ready threads. An FGMT processor switches threads every cycle and issue instructions from one thread. A CGMT processor is more likely to be built atop an in-order processor than an OOO processor because the in-order processor would normally stall the pipeline on a cache miss or branch. Although a FGMT processor can be implemented using an OOO processor as the baseline, it makes more sense to implement the FGMT processor based on an in-order processor since it only issues instructions from a single thread at one time. Current in-order multithreaded processors are usually implemented as CMT processors, such as Niagara, which is a combination of both CGMT and FGMT. Finally, SMT processors such as as Intel Nehalem [31] leverage OOO superscalar processors.

McPAT already has all the models for in-order and OOO base processors. Therefore, to model multithreaded processors McPAT also must model the sharing and duplication scheme for hardware resources as well as the extra hardware overhead for the type of multithreading needed. McPAT models multithreaded architectures based on designs of Niagara processors [25, 28], Intel hyperthreading technology [29], and early research in SMT architecture [57]. Specifically, it is critical to recognize the shared units, partitioned units, and private units for both CMT and SMT processors and model the architectural level models in McPAT accordingly.

Architecture	Private and Duplicated Units	Partitioned and Tagged Units	Share Units
SMT	Instruction buffers, RAS, Architecture RF, RAT (FRAT and RRAT), inter-stage buffers	ITLB, DTLB, BTB, BPT, ROB, instruction decoders, load buffers, store buffers,	Functional units, Icache, Dcache, L2cache, RS, instruction issue queue, physical register files
CMT	Instruction buffers, RAS, architecture RF, inter-stage buffers	ITLB, DTLB, BTB, BPT, instruction decoders, load buffers, store buffers,	Functional units, Icache, Dcache, L2cache,

Table 2: Sharing and duplication assumptions for hardware resources in McPAT for multithreaded processors

Table 2 shows the sharing and duplication assumptions for hardware resources in McPAT for multithreaded processors. All private units are modeled in a duplicated manner, with the number of copies being equal to the number of hardware threads. Partitioned units are modeled with extra thread ID in hardware. Each entry of the partitioned unit is assumed to include a CAM portion to store thread IDs as tags. Therefore some of these units, such as the branch predictor, are changed from RAM structures in single-threaded processors from cache-like structures with both tag and data arrays in multithreaded processors. Models of shared units in multithreaded are the same as that of single threaded processor models. Since McPAT computes energy per access to calculate final dynamic power, the duplicated units for non-active threads will not affect the dynamic power. However, these units will affect the leakage power and area of a multithreaded target design compares to a single threaded version.

5.2 Network on Chip (NoC)

An NoC has two main components: signal links and routers. For signal links between hops, we use hierarchical repeated wires, as described in Section 6. We use the same analytical approach as used in core modeling to model routers:

breaking the routers into basic building blocks such as flit buffers, arbiters, and crossbars; then building analytical models for each building block. McPAT models power, area and timing together for NoCs.

5.2.1 Routers

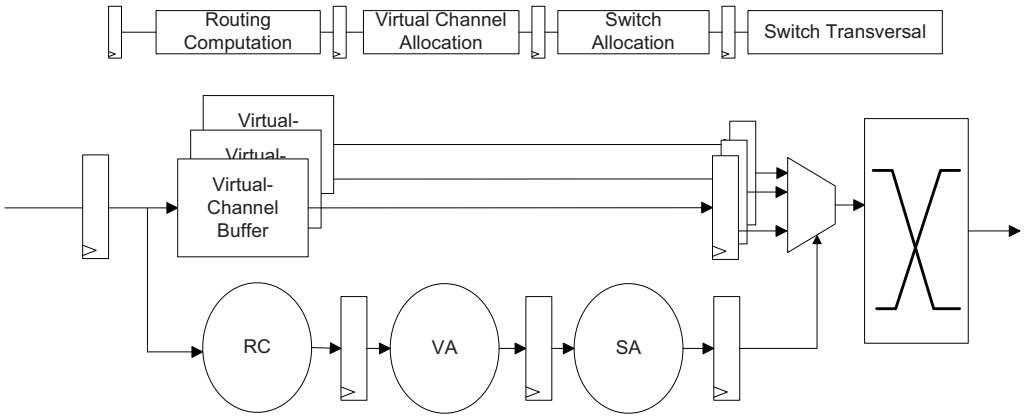


Figure 14: Router Model in McPAT

As shown in Figure 14, McPAT models a traditional router with four stages. Although innovative research such as speculation and look ahead can reduce the number of pipeline stages, we choose to model traditional routers as we believe they are more generic. The four stages of a traditional router are routing computation (RC), virtual channel allocation (VA), switch allocation (SA), and switch transversal (ST). Each stage involves different hardware components. At the RC stage, information from the head flit is used by the routing computation block to select an output port (or ports). At the same time, the arriving flits are stored in the virtual-channel flit buffer. During the VA stage, the result of the routing computation is sent to the virtual-channel allocator. If successful, the allocator assigns a single output virtual channel on one of output channels. At the SA stage, each active virtual channel bids for a time-slice of the switch so that the buffered flit on the virtual channel can transverse the switch to the output unit containing its output virtual channel. If successful the flit traverses the switch at the ST stage. Also during the SA stage, a flit is read out from the flit buffer so that it will be ready to transverse the switch when the grant signal is set. However, it is also possible that the flit does not get a grant signal. In order to avoid wasting read operations and to have the buffer read in parallel with the switch allocation, we assume that all virtual channel buffer with a grant signal from the virtual-channel allocator have one flit read out and stored in a flip flop based buffer. The grant signal from the SA is used to choose which buffer to send its flit into the switch. Depending on the flow control mechanism, not all flits will access all hardware in each stage. McPAT requires activity factors on each component to compute total dynamic energy. If the performance simulator can not provide default statistics, McPAT assumes wormhole routing, where only head flits will go through all the pipeline stages. The body flits will bypass hardware such as routing computation logic. The latency of each pipeline stage is computed as in Equations 23, 24, 25, 26, where $Delay_{RC}$, $Delay_{VA}$, $Delay_{SA}$ and $Delay_{crossbar}$ are the latency of logic at each stage, respectively. $T_{virtual-channel-buffer}$ is the access time of the virtual channel buffer. $T_{pipeline}$ is the delay of a register. The maximum clock frequency is shown in Equation 27

$$T_{RC} = Max(Delay_{RC}, T_{virtual-channel-buffer}) + T_{pipeline} \tag{23}$$

$$T_{VA} = Delay_{VA} + T_{pipeline} \tag{24}$$

$$T_{SA} = Max(Delay_{SA}, T_{virtual-channel-buffer}) + T_{pipeline} \tag{25}$$

$$T_{ST} = Delay_{crossbar} + T_{pipeline} \tag{26}$$

$$F_{router} = 1/Max(T_{RC}, T_{VA}, T_{SA}, T_{ST}) \tag{27}$$

5.2.1.1 Flit Buffer

The flit buffers are one of the most important structures in a router. Not all flits use routing computation logic in the RC stage, but all flits need to access the flit buffer. We model the flit buffer as a SRAM FIFO using the internal CACTI module. Both the access time and cycle time of the flit buffer must be less than the clock period of the router to meet timing constraints. One important design decision of flit buffer is its sharing scheme between virtual channels or physical channels. The flit size of modern router can be large, and the number of ports of modern router can be high. Therefore, implementing large shared buffers with large numbers of read/write ports puts very high pressure on the flit buffer to satisfy the timing constraint and to achieve reasonable area and power results. According to the analysis in [11], McPAT uses the one flit buffer per physical channel implementation with one output port per virtual channel by default as shown in Figure 15.

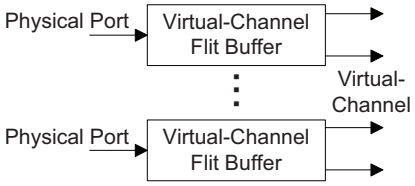


Figure 15: Flit Buffer organization Model in McPAT

5.2.1.2 Arbiter and Allocator

Both VA and SA stages need an allocator to map multiple requests to multiple available resources. An allocator performs matching between a group of resources and a group of requesters, and it can be implemented by two stages of arbiters. An arbiter assigns a single resource to one of a group of requesters. McPAT models a queueing arbiter and a matrix arbiter. Modeling a queuing arbiter is no different from modeling a RAM array. A matrix arbiter is modeled based on the circuit topology in [11, 59].

McPAT models both input-first and output-first separable allocators. A separable allocator performs allocation as two sets of arbitration: one across the inputs and one across the outputs. In an input-first separable allocator, an arbitration is first performed to select a single request at each input port. Then, the outputs of these input arbiters are sent to a set of output arbiters to select a single request for each output port. In contrast, in an output first separable allocator, output arbitration is performed first and then the input arbitration is performed. The total latency of the two stages of arbiters in the allocator must be less than the target clock period to satisfy the timing constraints.

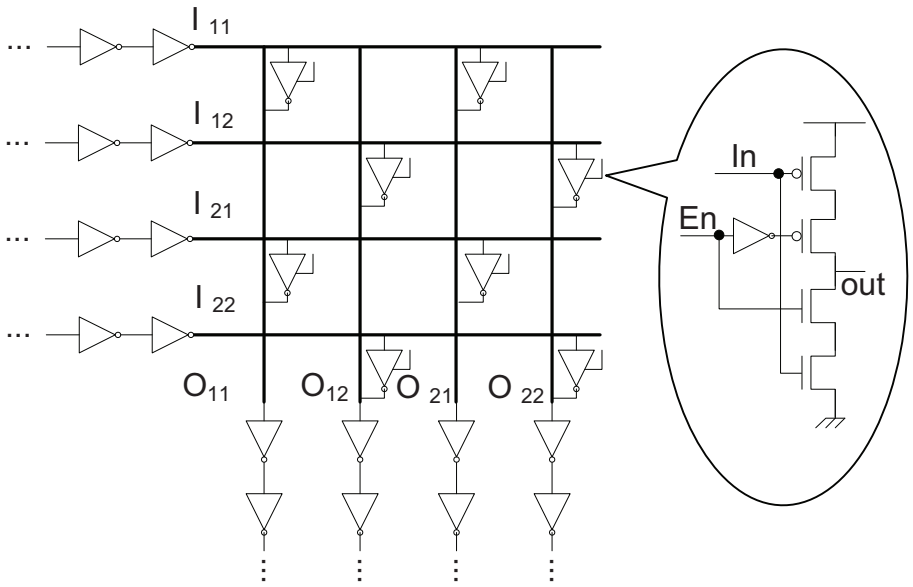


Figure 16: McPAT's crossbar model shown as a $2X2$ crossbar with data width of 2.

5.2.1.3 Crossbar

As shown in Figure 16, we model the matrix crossbar with the cross points being implemented as tri-state buffers. In a normal router architecture, although the whole router is pipelined we do not assume the crossbar itself is pipelined. The main reasons for not modeling pipelined crossbar switches are: (1) the repeated wires in the crossbar route in both horizontal and vertical directions which makes it very difficult to drop flip-flops underneath the wires. (2) only one cycle is allocated to switch transversal. Optimizing the crossbar is hard as there are multiple parameters need to be optimized: the sizing of repeaters in the metal wire, the signal wiring pitch of the wire array, the sizing of tri-state buffer, the sizing of input and output drivers. All these parameters depend on each other. McPAT first decides the signal wiring pitch. The tri-state buffers are then sized to have the same driveability as the optimized repeaters inside the wires of the crossbar. The tri-state buffers used in crossbars are shown in Figure 16. In order to provide same driveability, the size of PMOS and NMOS devices in the tri-state buffer are set to be double the size of PMOS and NMOS devices in the repeater. Then, input drivers and output drivers are sized using logical effort based on the gate load and the wire load of the "on" path in the crossbar.

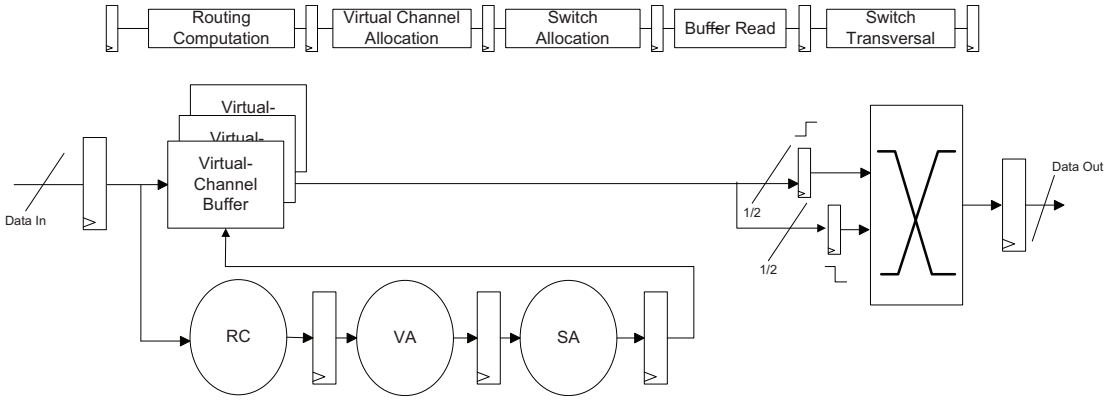


Figure 17: Router with double-pumped crossbar modeled in McPAT

McPAT also models a double-pumped crossbar [58] that reduces die area for on-chip interconnect intensive designs. McPAT assumes a five stage router when a double-pumped crossbar is used. The modeled router architecture with double pumped crossbar in McPAT is shown in Figure 17. The double-pumped crossbar requires a full clock cycle to use both positive and negative clock edges. Therefore, having virtual buffers read-ahead and using the SA grant signal to select data from multiple buffers will violate the timing requirements. The buffer read stage cannot be done in parallel with SA since otherwise the set-up time of first flip-flop in double-pumped crossbar cannot be satisfied. Unlike a single-pumped crossbar, a double pumped crossbar is a fully synchronized circuit. The delay of the double pumped crossbar is shown in Equation 28, which must be less than half clock cycle.

$$T_{xbar} = T_{M0} + T_{S0} + T_{Mux} + T_{xbarwire} \tag{28}$$

5.2.2 Inter-router Link

Inter-router links can consume 30% of total NoC power and need to be modeled carefully. McPAT assumes that all the inter-router links are repeated wires and appropriately pipelined when necessary. For the inter-router links, throughput is more important than latency since the links can be heavily pipelined. On the other hand, pipelining is not free as the flip-flops used in the pipelined occupy area and consume power. Therefore, McPAT first attempts satisfy the timing without adding flip-flops in the wire. Only when timing constraints cannot be satisfied, will McPAT insert flip-flops in the links. This is also reasonable because only those long links need to be placed in higher metal layer and be pipelined.

During optimization, McPAT first attempts to assign the links to different metal layers in the hierarchical wire model at the circuit level and computes the delays of the link at each metal layer. If the timing constraint can not be satisfied at the current metal layer, McPAT will vary the signal wiring pitch, use double or triple wide metals, and move to higher levels of metal, trying to satisfy the timing requirement. It is important to note that the links are modeled as repeated wires, therefore, repeater are included in all the computations. If all attempts are failed, McPAT assumes the links must be pipelined in order to satisfy the timing constraints. The number of pipeline stages are computed according to Equation 29, where N_{stage} is number of pipeline stages needed for the link, $Latency$ is the latency of the unpipelined link, T_{Clock} is the target clock period for the NoC, and $Throughput$ is the user defined throughput with regards to clock.

$$N_{stage} = (Latency/T_{Clock}) * Throughput \tag{29}$$

5.3 On-chip Caches

McPAT supports both private and shared/coherent caches. Private caches are models in the same way as instruction and data caches inside a core. It models coherent/shared caches by modeling the hardware that stores directory information associated with shared/coherent caches. Depending on the architecture, the hardware with directory information can be mapped to CAM structures at the circuit level as in Niagara processors [25,28] or directory cache structures as in Alpha 21364 [24]. The CAM implementation is usually used in shared cache structures, shadowing the tags of higher level cache to support quick lookup. However, this approach has high overhead since CAM is expensive and the size of the CAM is proportional to the total capacity of caches that share the same low level cache. The directory cache is designed as a dedicate cache for directory information. The complete information of the directory is assumed to be stored in main memory (or special memory section). This approach is usually used in coherent cache or last level shared cache.

5.4 Memory controller

McPAT models on-chip memory controllers. The Memory controller models follow a design from Denali [12] that contains three main hardware structures: 1) the frontend engine responsible for rescheduling the memory requests, 2) the transaction processing engine that has the logic and sequencer to generate the command, address, and data signal, and 3) the physical interface (PHY) that serves as an actual channel of the memory controller for communicating off-chip to memory.

5.4.1 Front-end engine

The front-end engine is responsible for reordering memory requests, buffering data, selecting memory requests, and sending them to the backend engine. There are three main components in the front-end engine: a request reorder buffer, a read request buffer, and a write request buffer. The request reorder buffer is mapped to a fully-associative structure in McPAT. Its CAM portion contains the memory request type, memory address, current page status in main memory, and memory channel ID. By associative searching in the request reorder buffer for each incoming memory request, the front-end can select appropriate memory requests based on a pre-defined scheduling policy. The data portion of the request reorder buffer contains an index of the memory requests that are stored in a read or write queue. It is possible that multiple memory requests are granted at the same time. Selection logic is needed to pick one request from the valid candidates. The selection logic is modeled in the same way as that used in wakeup logic in an instruction issue queue.

The read and write request buffers are modeled as RAM arrays. The data field of the buffers contains physical memory address and memory data. Although each memory operation issued from a last level cache request a whole cache block, we assume the memory controller supports a requested-word-first optimization [18].Hence, the data widths of the read and write request queues are equal to the sum of physical memory address, memory data bus width, and ECC bit width.

5.4.2 Transaction processing engine and PHY

The Transaction processing engine has the logic and sequencer to generate the command, address, and data signals, and is responsible for taking care of routine jobs such as refresh, ECC, and memory scrubbing. Modeling the transaction processing engine and PHY are difficult at an architectural level since they involve highly irregular logic, analog devices and transmission-lines. We empirically model PHY and the transaction processing engine according to published data from Rambus [33] and AMD [5].

5.5 Clocking

Clocking circuitry has two main parts: the phase-locked loop (PLL) with fractional dividers to generate the clock signals for multiple clock domains, and the clock distribution network to route the clock signals. McPAT uses an empirical model for the power of a PLL and fractional divider, by scaling published results from Sun and Intel [3, 45]. The clock distribution network can be directly mapped to the clock network model at the circuit level.

6 Circuit Level Modeling

6.1 Hierarchical repeated wires

Hierarchical repeated wires are used to model both local and global on-chip wires. Performance of wires is governed by two important parameters: resistance and capacitance. We model short wires using a one-section π-RC model [20]

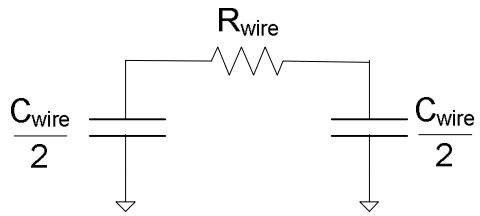


Figure 18: One-section π-RC model that we have assumed for plain wires.

as shown in Figure 18, where R_{wire} and C_{wire} for a wire of length L_{wire} are computed by Equations 30 and 31. We use Equations 32 to 34 from [20, 21] to compute $R_{unit-length-wire}$ and $C_{unit-length-wire}$ of a plain wire. Figure 19 shows the wire capacitance model used in Equations 32 to 34 from [20].

$$R_{wire} = L_{wire} R_{unit-length-wire} \tag{30}$$

$$C_{wire} = L_{wire} C_{unit-length-wire} \tag{31}$$

$$R_{unit-length-wire} = \alpha_{\text{scatter}} \frac{\rho}{(thickness - barrier - dishing)(width - 2 * barrier)} \tag{32}$$

(33)

$$C_{unit-length-wire} = \varepsilon_0 (2M\varepsilon_{\text{horiz}} \frac{thickness}{spacing} + 2\varepsilon_{\text{vert}} \frac{width}{ILD_{\text{thick}}}) + fringe(\varepsilon_{\text{horiz}}, \varepsilon_{\text{vert}}) \tag{34}$$

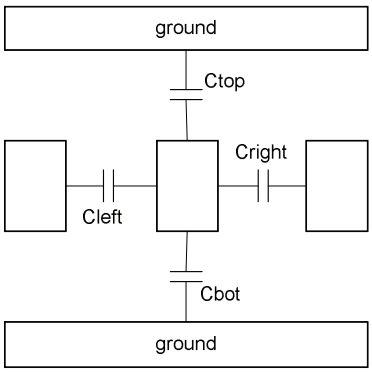


Figure 19: Capacitance model from [20].

Wires scale more slowly than gates with respect to RC delays, and unbuffered wires cannot keep up with the improved transistor delay. For long wires, we use a repeated wire model [20] with optimized repeaters as in CACTI 6.0 [38]. The equations used to find optimal sizing of the repeaters are reproduced below, where c_0, c_p and r_s are constants for a given technology.

$$L_{optimal} = \sqrt{\frac{2r_s(c_0 + c_p)}{R_{unit-length-wire}C_{unit-length-wire}}} \tag{35}$$

$$\tag{36}$$

$$S_{optimal} = \sqrt{\frac{r_s C_{unit-length-wire}}{R_{unit-length-wire}c_0}} \tag{37}$$

We assume multiple metal planes for local, intermediate, and global interconnect, each with different wire pitches and aspect ratios. Our hierarchical wire model assumes ten metal planes, with four layers for local interconnect, and two each for intermediate, semi-global, and global interconnect.

Wires on different planes have different pitches and aspect ratios. The assignment of signals to wiring planes plays a key role in determining power, area, and timing characteristics. McPAT's optimizer automatically assigns wires to planes to achieve specified objectives by trying different metal layers and varying effective wiring pitches. First, our model optimizes the buffers for each wiring plane. Then, the delays of buffered wires are calculated for the first assigned layer in the wiring plane hierarchy. If the current wiring layer cannot satisfy the delay constraints, the wires are moved to a higher wiring plane for lower latency, at the expense of a higher power and area penalty. The effective wiring pitches on semi-global and global planes are also increased to satisfy the target latency as described in [32]. Latches are also inserted and modeled when necessary to satisfy the target clock rate as mentioned in Section 5.2.2

6.2 Arrays

McPAT includes models for three basic array models at the circuit level: RAM-, CAM-, and DFF-based arrays. The models for these structures are based on CACTI [55], with several important extensions. First, McPAT re-implements the CAM model from CACTI to more accurately reflect its multi-banked/ported structure and also adds a write operation model and a non-associative-search read model. Second, McPAT adds a detailed DFF array model. Finally, McPAT adds gate leakage and improved timing models for arrays, such as a CAM cycle time model. As mentioned in Section 3.4, the details of the array modeling methods can be found in technical report of CACTI 7.0 [1].

6.3 Logic

McPAT employs three different schemes for modeling logic blocks, depending on the complexity of the block. For highly regular blocks with predictable structures, such as memories or networks, McPAT uses the algorithmic approach of CACTI [55]. For structures that are less regular but can still be parameterized, such as thread selection or decoding logic, McPAT uses analytic models similar to those in [43] as explained in Section 5. Finally, for highly customized blocks such as functional units, McPAT uses empirical models based on published data for existing designs scaled to different technologies. For example, the ALU and FPU models are based on actual designs by Intel [36] and Sun [34].

6.4 Clock distribution network

A clock distribution network is responsible for routing clock signals of different frequencies to clock domains, with drops to individual circuit blocks. It is a special case of a hierarchical repeated wire network. We model it separately from the normal repeated wires not only because it has strict timing requirements with large fanout loads spanning the entire chip but also because it is wave-pipelined [60] that is completely different from normally pipelined global interconnects. It consumes a significant fraction of total chip power [16]. We represent a clock distribution network

using a separate circuit model that has three distinct levels: global, domain, and local. We assume an H-tree topology for global-level and domain-level networks and a grid topology for the local networks as shown in both the Niagara processors [28, 41] and the Intel Itanium processors [13, 35]. We assume the global-level clock distribution network can only be implemented using global metal layers. The domain-level and local clock distribution network can be implemented using both semi-global and global metal layers. NAND gates are used at all final clock heads to enable clock gating.

7 Technology Level Modeling

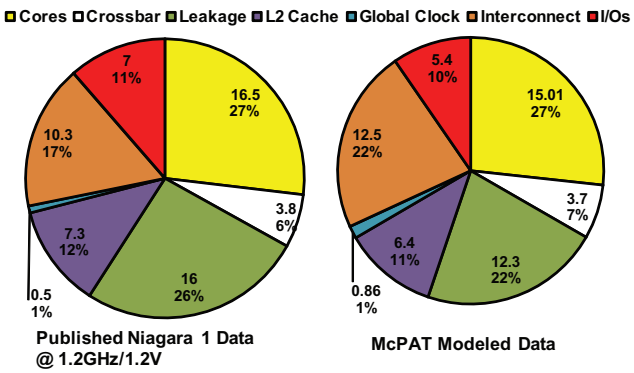
Following the same methodology as in CACTI5 [56], McPAT uses MASTAR [50] to derive device parameters from the ITRS roadmap [50] and R. Ho's work [20, 21] to derive wire parameters for different technology nodes. The current implementation of McPAT includes data for the 90nm, 65nm, 45nm, 32nm, and 22nm technology nodes, which covers the ITRS roadmap through 2016. The ITRS assumes that planar bulk CMOS device will reach practical scaling limits at 36nm, at which point the technology will switch to silicon-on-insulator (SOI). Below 25nm, the ITRS predicts that SOI will reach its limits and that double-gate devices will be the only option. McPAT captures each of these options, making it scalable with the ITRS roadmap. McPAT shares the same technology level modeling with CACTI 5. The main improvements at technology level modeling are: (1) the technology generations have been extended to 22nm with double gate (DG) technology; (2) the interconnect also has been extended to 22nm technology. (3) double/triple/quad-width wires area added into the interconnect technology model.

8 Validation

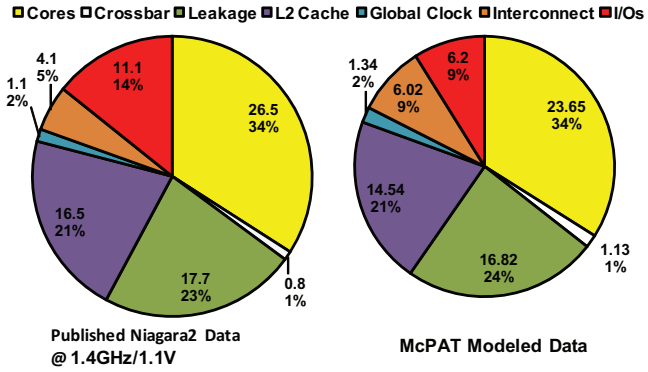
The main focus of McPAT is accurate power and area modeling at the architectural level when timing is given as a main design constraint. Both *relative* and *absolute* accuracy are important for architectural level power modeling. Relative accuracy means that changes in modeled power as a result of architecture modifications should reflect on a relative scale the changes one would see in a real design. While relative accuracy is critical, absolute accuracy is also important if one wants to compare results against thermal design power (TDP) limits, or to put core power savings in the context of the whole processor or whole system.

We compare the output of McPAT against published data for the 90nm Niagara processor [34] running at 1.2GHz with a 1.2V power supply, the 65nm Niagara2 processor [41] running at 1.4GHz with a 1.1V power supply, the 65nm Xeon processor [45] running at 3.4GHz with a 1.25V power supply, and the 180nm Alpha 21364 processor [24] running at 1.2GHz with a 1.5V power supply. There are in-order and out-of-order processors as well as single-threaded and multithreaded processors in the validation targets. Therefore, the validations stress McPAT in a comprehensive and detailed way. The comparisons against Niagara and Niagara2 processors also test our ability to cross technology generations and retain accuracy. The configurations for the validations are based on published data on the target processors in [24, 34, 41, 45, 54], including target clock rate, working temperature, and architectural parameters. The target clock rate is used as the lower bound for the timing constraint that is used in McPAT to determine the basic circuit property and must be satisfied before other optimizations and trade-offs can be applied. Therefore, the generated results must match the clock rate of the actual target processor. Because timing (target clock rate) is already considered when computing and optimizing power and area, only power and area validation results are shown in this section.

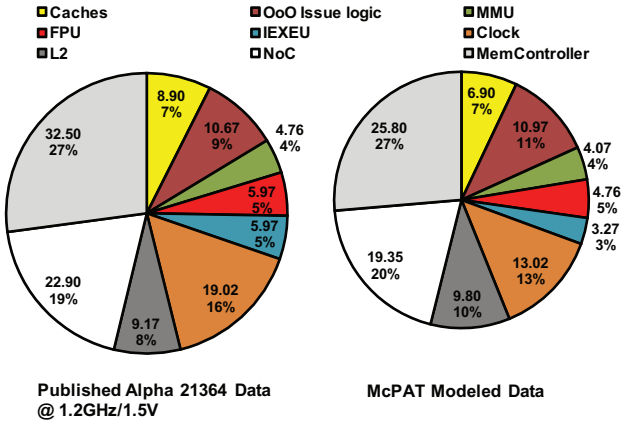
Figure 20 shows these results. Unfortunately, the best power numbers we have for these processors are for peak power rather than average power. Fortunately, McPAT can also output peak power numbers based on maximum activity



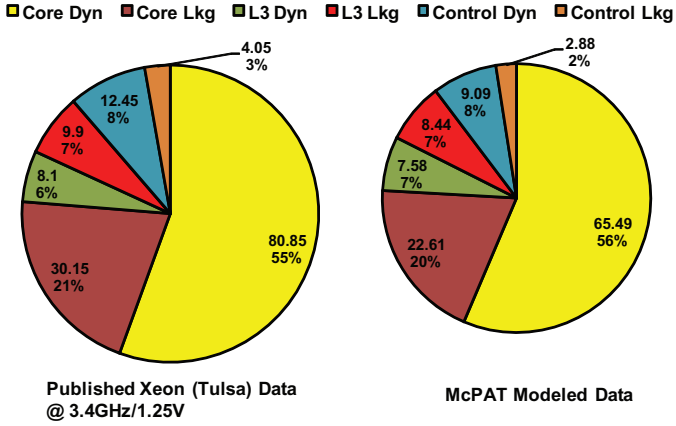
(a) Validation against Niagara processor.



(b) Validation against Niagara2 processor.



(c) Validation against Alpha 21364 processor.



(d) Validation against Xeon Tulsa processor.

Figure 20: McPAT validation results. The numbers in all charts are the reported and modeled power numbers of the components. The percentages denote the ratios of the component power to the total power. There are miscellaneous components such as SoC logic and I/O that McPAT does not model in detail because their structures are unknown. Therefore, 61.4W out of the total 63W are modeled for Niagara, 77.9W out of the total 84W are modeled for Niagara2, 119.8W out of the total 125W are modeled for Alpha 21364, and 145.5W out of the total 150W are modeled for Xeon Tulsa.

factors and maximum switching activity. Results show that modeled power numbers track the published numbers well. For the Niagara processor, the absolute power numbers for cores and crossbars generated by McPAT match very well with the published data. Over all seven components, the average difference in absolute power between the modeled power numbers and published Niagara data is just 1.47W, for an average error per component of 23%. That number seems high, but the two big contributors are clock power (72% error, but a small contributor to total power) and leakage power (23% error). Both are significantly more accurate for the Niagara2. For Niagara2, the average error is 1.87W (26%), but by far the biggest contributor (4.9W error) is the I/O power. This arises because Niagara2 is a complicated SOC with different types of I/Os including memory, PCI-e, and 10Gb Ethernet [41] while McPAT only models on-chip memory controllers/channels as I/Os. If we were measuring average power instead of peak power, this difference would shrink given the expected activity factors of those components. The modeled power number of the OOO issue logic, key component of the OOO core in the Alpha 21364 processor, is very close to the reported power with only 2.78% difference. Although there are no detailed power breakdowns of both the core and uncore parts of Xeon Tulsa, the modeled bulk power of core and uncore comes close to reported data, with -20.63% and -11% respectively.

Table 3 shows the comparison of total power for validations against the target processors. Differences between the total peak power generated by McPAT and reported data are 10.84%, 17.02%, 21.68%, and 22.61% for Niagara,

Processor	Published total power	Modeled total power	McPAT results	% McPAT error
Niagara	63W	61.4W	56.17W	-10.84
Niagara2	84W	77.9W	69.70W	-17.02
Alpha 21364	125W	119.8W	97.9W	-21.68
Xeon Tulsa	150W	145.5W	116.08W	-22.61

Table 3: Validation results of McPAT with regard to total power of target processors.

Processor	Published die size (mm^2)	McPAT results (mm^2)	McPAT error %
Niagara	378	295	-21.8
Niagara2	342	248	-27.3
Alpha 21364	396	324	-18.2
Xeon Tulsa	435	362	-16.7

Table 4: Validation Results of McPAT with regard to chip die area of target processors.

Niagara2, Alpha 21364, and Xeon Tulsa, respectively. It is worth noting that these differences include the unknown parts that we do not model in detail. Chip-to-chip power variation in recent microprocessor designs [9] is also comparable to the errors reported in Table 3.

Table 4 compares the published die sizes of the target processors with the McPAT results, which shows that the modeled area numbers track the published numbers well. The error is higher for Niagara2 because it has more types of I/O components than others as mentioned above, which are not modeled in McPAT. It is important to point out that even if we use empirical models for highly irregular logic, we still see a reasonable match between McPAT results and reported data. The area validation of Alpha is a good example. Although we do not use Alpha data when building our empirical models, the area validation of Alpha 21364 is only 18.2% off from the reported data. This shows that both the analytical models and empirical models for area modeling have good accuracy. Given the generic nature of McPAT, we consider these errors on both power and area acceptable.

References

[1] "CACTI7.0 Technical Report," Tech. Rep., To Be released.

[2] K. Agarwal, H. Deogun, D. Sylvester, and K. Nowka, "Power Gating with Multiple Sleep Modes," *ISQED*, 2006.

[3] H.-T. Ahn and D. Allstot, "A Low-jitter 1.9-V CMOS PLL for UltraSPARC Microprocessor Applications," *JSSC*, vol. 35, no. 3, pp. 450–454, 2000.

[4] H. Al-Hertani, D. Al-Khalili, and C. Rozon, "UDSM subthreshold leakage model for NMOS transistor stacks," *Microelectron. J.*, vol. 39, no. 12, pp. 1809–1816, 2008.

[5] AMD, "AMD Opteron Processor Benchmarking for Clustered Systems," *AMD WhitePaper*, 2003.

[6] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," *Computer*, vol. 35, no. 2, 2002.

[7] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The M5 Simulator: Modeling Networked Systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, 2006.

[8] B. Bishop, T. P. Kelliher, and M. J. Irwin, "The Design of a Register Renaming Unit," in *Great Lakes VLSI '99*, 1999.

[9] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, "Parameter Variations and Impact on Circuits and Microarchitecture," in *DAC*, 2003.

[10] J. Chang, *et al.*, "The 65-nm 16-MB Shared On-Die L3 Cache for the Dual-Core Intel Xeon Processor 7100 Series," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 4, Apr 2007.

[11] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, 2003.

[12] Denali, "Using Configurable Memory Controller Design IP with Encounter RTL Complier," *Cadence CDNLive!*, 2007.

[13] T. Fischer, F. Anderson, B. Patella, and S. Naffziger, "A 90 nm variable frequency clock system for a power-managed itanium architecture processor," in *ISSCC*, Feb. 2005, pp. 294–295.

[14] V. George, *et al.*, "Penryn: 45-nm next generation Intel core 2 processor," in *ASSCC'07 IEEE Asian Solid-State Circuits Conference*, 2007.

[15] A. D. Gloria and M. Olivieri, "An application specific multi-port RAM cell circuit for register renaming units in high speed microprocessors," in *ISCAS*, 2001.

[16] M. K. Gowan, L. L. Biro, and D. B. Jackson, "Power Considerations in the Design of the Alpha 21264 Microprocessor," in *DAC*, 1998.

[17] S. Gupta, S. Keckler, and D. Burger, "Technology Independent Area and Delay Estimates for Microprocessor Building Blocks," UT Austin, Department of Computer Science," Tech. Rep., 2000.

[18] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach. 4th Edition*, 2007.

[19] G. Hinton, D. Sager, M. Upton, D. Boggs, D. P. Group, and I. Corp, "The Microarchitecture of the Pentium 4 Processor," *Intel Technology Journal*, vol. 1, 2001.

[20] R. Ho, "On-chip Wires: Scaling and Efficiency," Ph.D. dissertation, Stanford University, 2003.

[21] M. Horowitz, R. Ho, and K. Mai, "The future of wires," In Invited Workshop Paper for SRC Conference., Tech. Rep., 1999, available at http://velox.stanford.edu/.

[22] Intel, "P6 Family of Processors Hardware Developer's Manual," *Intel White Paper*, 1998.

[23] Ir. Frank Vanden Berghen, "XML Parser," http://www.applied-mathematics.net/tools/xmlParser.html.

[24] A. Jain, *et al.*, "A 1.2 GHz Alpha Microprocessor with 44.8 GB/s Chip Pin Bandwidth," in *ISSCC*, 2001.

[25] T. Johnson and U. Nawathe, "An 8-core, 64-thread, 64-bit Power Efficient Sparc SoC (Niagara2)," in *ISPD*, 2007.

[26] R. E. Kessler, "The Alpha 21264 Microprocessor," *IEEE Micro*, vol. 19, no. 2, 1999.

[27] N. S. Kim, *et al.*, "Leakage Current: Moore's Law Meets Static Power," *Computer*, vol. 36, no. 12, 2003.

[28] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-Way Multithreaded Sparc Processor," *IEEE Micro*, vol. 25, no. 2, 2005.

[29] D. Koufaty and D. T. Marr, "Hyperthreading Technology in the Netburst Microarchitecture," *IEEE Micro*, vol. 23, no. 2, 2003.

[30] D. Kroft, "Lockup-free Instruction Fetch/Prefetch Cache Organization," in *ISCA*, 1981.

[31] R. Kumar and G. Hinton, "A family of 45nm IA processors," *ISSCC*, pp. 58–59, 2009.

[32] R. Kumar, V. Zyuban, and D. M. Tullsen, "Interconnections in Multi-Core Architectures: Understanding Mechanisms, Overheads and Scaling," in *ISCA*, 2005.

[33] H. Lee, *et al.*, "A 16Gb/s/link, 64GB/s Bidirectional Asymmetric Memory Interface," *JSSC*, vol. 44, no. 4, 2009.

[34] A. S. Leon, K. W. Tam, J. L. Shin, D. Weisner, and F. Schumacher, "A Power-Efficient High-Throughput 32-Thread SPARC Processor," *JSSC*, vol. 42, 2007.

[35] P. Mahoney, E. Fetzer, B. Doyle, and S. Naffziger, "Clock Distribution on a Dual-Core Multi-Threaded Itanium Family Processor," in *ISSCC*, 2005.

[36] S. Mathew, M. Anders, B. Bloechel, T. Nguyen, R. Krishnamurthy, and S. Borkar, "A 4-GHz 300-mW 64-bit Integer Execution ALU with Dual Supply Voltages in 90-nm CMOS," *JSSC*, vol. 40, no. 1, 2005.

[37] K.-S. Min, K. Kanda, and T. Sakurai, "Row-by-row dynamic source-line voltage control (RRDSV) scheme for two orders of magnitude leakage current reduction of sub-1-V-VDD SRAM's," in *ISLPED '03*, 2003.

[38] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A Tool to Model Large Caches," School of Computing, University of Utah, Tech. Rep., 2007.

[39] S. Narendra, V. De, D. Antoniadis, A. Chandrakasan, and S. Borkar, "Scaling of stack effect and its application for leakage reduction," in *ISLPED '01*, 2001.

[40] A. Naveh, E. Rotem, A. Mendelson, S. Gochman, R. Chabukswar, K. Krishnan, and A. Kumar, "Power and Thermal Management in the Intel Core Duo Processor," *Intel Technology Journal*, vol. 10, pp. 109–122, 2006.

[41] U. Nawathe, M. Hassan, K. Yen, A. Kumar, A. Ramachandran, and D. Greenhill, "Implementation of an 8-Core, 64-Thread, Power-Efficient SPARC Server on a Chip," *JSSC*, vol. 43, no. 1, 2008.

[42] K. Nose and T. Sakurai, "Analysis and Future Trend of Short-circuit Power," *IEEE TCAD*, vol. 19, no. 9, 2000.

[43] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Complexity-Effective Superscalar Processors," in *ISCA*, 1997.

[44] A. F. Rodrigues, "Parametric Sizing for Processors," Sandia National Laboratories," Tech. Rep., 2007.

[45] S. Rusu, S. Tam, H. Muljono, D. Ayers, and J. Chang, "A Dual-Core Multi-Threaded Xeon Processor with 16MB L3 Cache," in *ISSCC*, 2006.

[46] E. Safi, P. Akl, A. Moshovos, A. Veneris, and A. Arapoyianni, "On the Latency, Energy and Area of Checkpointed, Superscalar Register Alias Tables," in *ISLPED*, 2007.

[47] N. Sakran, M. Yuffe, M. Mehalel, J. Doweck, E. Knoll, and A. Kovacs, "The Implementation of the 65nm Dual-Core 64b Merom Processor," in *ISSCC'07*, 2007, pp. 106–590.

[48] J. Schutz and C. Webb, "A scalable x86 cpu design for 90 nm process," in *ISSCC'04*, 2004, p. 62.

[49] Selantek, Tech. Rep., 2007.

[50] Semiconductor Industries Association, "Model for Assessment of CMOS Technologies and Roadmaps (MASTAR)," 2007, http://www.itrs.net/models.html.

[51] J. P. Shen and M. Lipasti, *Modern Processor Design: Fundamentals of Superscalar Processors*, 2004.

[52] J. Silc, B. Robic, and T. Ungerer, *Processor Architecture: From Dataflow to Superscalar and Beyond*, 1999.

[53] D. Sima, "The design space of register renaming techniques," *IEEE Micro*, vol. 20, no. 5, pp. 70–83, 2000.

[54] Sun Microsystems, "OpenSPARC," http://www.opensparc.net.

[55] S. Thoziyoor, J. Ahn, M. Monchiero, J. Brockman, and N. Jouppi, "A Comprehensive Memory Modeling Tool and its Application to the Design and Analysis of Future Memory Hierarchies," in *ISCA*, 2008.

[56] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "CACTI 5.1," HP Labs, Tech. Rep. HPL-2008-20.

[57] D. M. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, and R. L. Stamm, "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," in *ISCA*, 1996.

[58] S. Vangal, N. Borkar, and A. Alvandpour, "A Six-port 57GB/s Double-pumped Nonblocking Router Core," in *VLSI*, June 2005.

[59] H. Wang, X. Zhu, L.-S. Peh, and S. Malik, "Orion: A Power-Performance Simulator for Interconnection Networks," in *MICRO*, Nov 2002.

[60] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective. 3rd Edition*, 2004.

[61] K. C. Yeager, "The MIPS R10000 Superscalar Microprocessor," *IEEE Micro*, vol. 16, no. 2, 1996.

[62] V. Zaccaria, D. Sciuto, and C. Silvano, *Power Estimation and Optimization Methodologies for VLIW-Based Embedded Systems*. Norwell, MA, USA: Kluwer Academic Publishers, 2003.

[63] K. Zhang, *et al.*, "SRAM Design on 65-nm CMOS Technology with Dynamic Sleep Transistor for Leakage Reduction," *JSSC*, vol. 40, no. 4, pp. 895–901, 2005.