Energy Characterization of a Tiled Architecture Processor with On-Chip Networks

Jason Sungtae Kim, Michael Bedford Taylor, Jason Miller, David Wentzlaff
MIT Laboratory for Computer Science
{jasonkim, mtaylor, jasonm, wentzlaf} @ cag.lcs.mit.edu

ABSTRACT

Tiled architectures provide a paradigm for designers to turn silicon resources into processors with burgeoning quantities of programmable functional units and memories. The architecture has a dual responsibility: first, it must expose these resources in a way that is programmable. Second, it needs to manage the power associated with such resources.

We present the power management facilities of the 16-tile Raw microprocessor. This design selectively turns on and off 48 SRAM macros, 96 functional unit clusters, 32 fetch units, and over 250 unique processor pipeline stages, all according to the needs of the computation and environment at hand.

Categories and Subject Descriptors
C.1.4 [**Parallel Architectures**]: Distributed Architectures

General Terms
Design, Measurement, Performance

Keywords
Raw Microprocessor, Tile, Power, Scalar Operand Network

1. INTRODUCTION

The rapid shrinking of VLSI feature sizes brings the promise of both increasing frequencies and larger numbers of general-purpose functional units for high-performance microprocessors. The harsh realities of interconnect delay and power consumption seriously challenge the ability of microprocessor designers to fulfill these promises. A case in point is the Itanium 2 processor, which consumes 130 Watts and sports a zero-cycle fully-bypassed 6-way issue integer execution core. Despite occupying less than two percent of the processor die, this execution core spends half of its critical path in the bypass paths between the ALUs.

Current wide-issue processor designs rely on huge, global, centralized structures such as associative instruction windows, register renamers, heavily-ported register files, zero-cycle bypassed ALU clusters, and heavily-ported caches. As these structures scale up in size, internal interconnect delay becomes the dominant factor. This delay is due to increasing wire lengths, capacitive loads and effective logic levels. The desire to minimize delay keeps these structures centralized, and compounds the power distribution problem, because it clusters like-components together. This all but ensures that the high power consumption components will be situated near to each other. Moreover, architectural features required to extract parallelism from serial instruction streams require substantially more logic, resulting in reduced energy efficiency.

Microprocessor designers have recently found that the exponentially rising thermal dissipation of high performance microprocessors has begun to affect these systems' applicability even in non-mobile environments. High thermal dissipation precludes the manufacture of 1U rack-mount systems that contain early Itanium II or Pentium 4 processors. Heat production also affects the cost of these systems: the price of an appropriate cooling solution increases drastically with rising microprocessor temperatures. These power concerns have lead the Pentium 4 architects to include a thermal monitor and a mechanism to stop the processor's clock in order to prevent overheating [5].

If this trend continues, it appears as if all processors, whether high-performance or mobile, will soon need to be, in some sense of the term, "low-power."

2. TILED ARCHITECTURES

Growing interest in scalable alternatives to current centralized microprocessor designs has led a number of architectural researchers to propose architectures that organize silicon resources in more effective ways. These distributed, general-purpose, parallel, architectures have the common theme that they are composed out of a number of tiles, i.e., replicated processing elements containing functional unit, memory, and sequencing resources. These tiles are interconnected by generalized on-chip point-to-point multi-hop interconnection networks, and are registered on input. Larger processors are created by stamping out more tiles. Because these networks pipeline the inherent delay of routing signals between remote functional units, there are no global wires, and these architectures can attain high frequencies. Furthermore, the inherent wire delay is exposed, so that intelligent algorithms (whether compile-time or run-time) can distribute instructions across the remote tiles according to the sensitivity of the program to latency.

Because these tiled architectures are designed to manage large quantities of compute resources, good energy management is paramount. Fortunately, tiled designs have many advantages. First, the decentralized nature of these processors allows them to be scaled without increasing the number of ports on their register files, caches, or bypass networks. This greatly reduces the logic overhead per functional unit. Second, these architectures reflect the parallelism inherent in VLSI structures and do not require complex logic to extract parallelism from sequential instruction streams. Third, tiled designs provide easy boundaries for powering down or clock-gating unused register files, ALUs, cache banks, or fetch bandwidth, according to the needs of the program. Fourth, tiled designs are very regular and distribute power needs evenly across the die to prevent hotspots. Finally, the only long wires are the network links, which are structured and regular and can be tuned for low power. [2][4]

One such tiled architecture is the MIT Raw microprocessor, which divides the silicon area of a chip into a two-dimensional mesh

ISLPED'03, August 25–27, 2003, Seoul, Korea.

of 16 programmable tiles and connects them through on-chip, point-to-point networks. This processor demonstrates its allure through its ability to issue 16 different FPU, integer, load, store, or branch instructions each cycle. It also has 512 registers, and 2MB of distributed L1 cache. All of these numbers are larger than those of the Itanium II, which has a larger die. Other tiled designs include MIT's SCALE, UT Austin's GRID, Wisconsin's Multiscalar and ILDP, and Stanford's Smart Memories.

In this paper, we characterize the energy profile of the distributed 16-tile MIT Raw microprocessor. We start by describing Raw's architectural approach to power management for distributed processing resources. We describe the limited steps we took in the implementation of the processor to reduce power. We continue by characterizing the actual Raw chip's power usage according to the resources used. This data would be useful in particular to other architects who are interested in designing reduced-power distributed, tiled architectures. Finally, along the way, we touch on some ideas on how the power could be further reduced in the system.

3. THE RAW MICROPROCESSOR

The Raw microprocessor was fabricated using the 1.8V IBM SA-27E 0.15um 6-level Cu ASIC process. The chip core runs at 420 MHz at room temperature and nominal voltage, and reaches 500 MHz with increased supply voltage. This compares favorably with IBM-implemented PowerPCs in the same process.

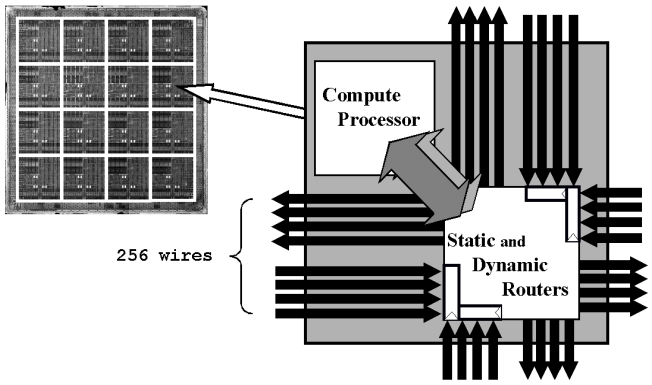

Figure 1: Raw microprocessor die photo and tile diagram

As shown in Figure 1, the 331 mm^2 chip contains 16 tiles arranged in a 4 by 4 mesh. The tiles are connected by four 32b point-to-point, on-chip, pipelined, mesh interconnection networks: two static and two dynamic. Each tile contains a single issue 8-stage in-order MIPS-style processor (called the "compute processor") that is coupled via register-mapped ports to three routers -- two dynamic, and one static. The tiles' 3-stage dynamic routers collectively control the dynamic, dimension-ordered, wormhole-routed on-chip interconnection network. The tiles' 5-stage static routers collectively control the two static networks. Each static router sequences a 64b instruction from an 8Kx64b instruction cache. Each instruction specifies a branch and up to 13 simultaneous routes that are to be performed on the static network. The static networks enable the output of an ALU in one tile to be routed, in-order, to the input of an ALU in a neighbor tile in just 3 cycles. The compute processor has a 4-stage FPU, an 8Kx32b instruction cache, and a 2-way set-associative 8Kx32 data cache. Routes off the edges of the networks are multiplexed down onto approximately 1080 pins. A more detailed description of the Raw processor can be found in [1][3].

We did not intend to implement any power-saving features at all in the processor, as the processor design fits very easily within the 80 W desktop-system power-envelope that IBM recommend for our package. Despite this, we discovered that the tiling approach yielded a very elegant method for power management, and as a result, we decided that the minimal amount of effort required was well worth it. The tiles discretize the silicon into easy-to-control regions; in this implementation, one tile represents 1/16 of the power in the chip. When we enter a phase of a computation that is very parallel, we wake-up enough tiles to efficiently exploit the parallelism, then when we return to a more serial phase that cannot benefit from the extra parallelism, we can put the tiles back to sleep.

Our efforts to reduce power consumption in the Raw processor focused around a "toggle-suppression" strategy: we ensure that wires in the system do not toggle unless they are actually computing something useful. The most important case to handle is when a tile's routers or compute processor stalls. In this case, the inputs to the functional units or crossbars are frozen, and the memory clocks are gated to prevent unnecessary pre-charging of the sense amplifiers.

Due to toggle-suppression, compute processors and routers consume minimal non-clock power when they are blocked waiting for messages on the network. Thus, in the serial phase of a computation, unused tiles have only to read a value that is not yet available from the network. This will cause a stall, and the pipeline will cease to toggle until the input to the above-mentioned parallel phase is received from the tile finishing the serial phase. The compute processor and routers wake up instantaneously when the data word is available, and sleep instantly when the next word is requested but is not available.

We also employed a toggle-suppression strategy for individual functional units and memories. Functional units in each compute processor were divided into six clusters. Each cluster's input register is individually enabled based on the needs of the instruction being issued. Clock inputs to memories are gated whenever practical.

Another power-saving feature that we considered was the idea of clock-gating at the tile level. Doing so would further reduce power for inactive tiles. This would have required a separation of the clock nets of the input FIFOs in the tiles, so that they could continue to receive data even as the rest of the tile was asleep. The valid bits of these FIFOs could be used to awaken the appropriate resources within the tile when a word arrives. We decided to avoid the complexities of this approach for the first version of the chip.

All told, we added a handful of control signals, multiplexers and registers to the Verilog source. A minimal amount of effort enabled the Raw microprocessor to selectively turn on and off 48 SRAM macros, 96 functional unit clusters, 32 fetch units, and over 250 unique processor pipeline stages, all according to the needs of the computation and environment at hand.

4. EXPERIMENTAL SETUP

The Raw motherboard could be easily mistaken for a PC motherboard; it has PCI slots, DIMM slots, and other standard PC connectors. It even fits in a PC case. The motherboard design separates out the power supplies for various components of the board; in particular, the chip core has a dedicated 1.8V current source, which allows us to measure the current consumed by the core while it is executing programs. The experimental data was collected with a core voltage of 1.8V, a core frequency of 100 MHz, and at room temperature. The corresponding values at higher voltages and frequencies can be easily calculated.

5. ENERGY CHARACTERIZATION

For this section, we used an ammeter connected to the Raw core power supply in order to measure power usage. All numbers were measured or derived from measurements. In all cases, we ran the measured item in unrolled loops to get an average number over time. We also replicated code across the tiles to increase the measurement magnitude. In many cases, we tried to make graphs more interesting by breaking aggregate numbers into constituent parts. This was accomplished by running sub-components of the program with the same data patterns. All in all, we found measurement at this level to be an enjoyable but tricky problem.

5.1 Idle State

When the clock is grounded, the chip draws a leakage current of 28 mA and dissipates 45 mW. When the Raw microprocessor's compute processors and routers are all stalled, the core consumes 1,271 mA. Using Synopsys CoverMeter, we verified that the only significant state elements that transition in this mode are two increment-by-1 cycle counters per tile and a single 77-bit serial shift register used for test apparatus. With the power for this logic well under 20 mA, we estimate that the measured number is no more than 2% off of the actual current due to the clock. See Table 1.

Table 1: Idle Current & Power: Leakage and Clock, 1.8V

Leakage@ Room Temp	28 mA	45 mW
Clock @ 100 MHz	1,271 mA	2,288 mW

5.2 Maximum, Typical, and Low Power

We start our investigation by examining the average current of three applications, as shown in Figure 2. The first application uses a single Raw tile with an average instruction mix. Ignoring clock power, it consumes a tiny fraction of the maximum power. This makes it clear that implementing clock-gating at the tile level is highly desirable for tiled architectures. The second application represents a typical highly-parallel application, vpenta, running with 16-tiles, which attains a 13x speedup over a single-tile version of the same application. Notice in particular, how the tiled approach, ignoring the clock power, provides a wide menu of power/performance points, even without modifying voltage or frequency. The last application is a synthetic application, created through experimentation, that maximizes power consumption. It consists of three independent parallel parts, running on all 16 tiles.

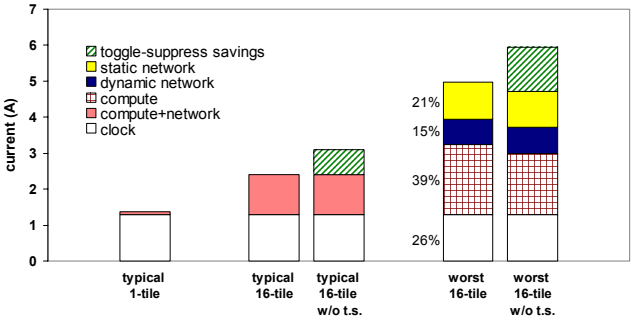

Figure 2: Average Current for Three Applications

The first part of the synthetic application executes load instructions in an unrolled loop. It alternates between loading from an address with offset 0xFFFC and an address with offset 0. These addresses contain the values 0 and 0xffffFFFF. This is the most power-consuming instruction sequence on the compute processor. The second part routes alternating sequences of 0 and 0xffffFFFF across 128 network links on the static network. This includes a total of 96 words routed each cycle across the inter-tile 4 mm wires. Maximum power on the dynamic networks is achieved by launching special 4-word "tail-chasing messages" that route themselves indefinitely in a circle inside the network. They consist of a header word and a data payload that selectively inverts and un-inverts the header word to create maximal toggles. This routes another 96 words over 4 mm inter-tile links each cycle. Loop-back devices on the I/Os would enable another 28 words per cycle of 4 mm routes.

The bars in the graph labeled "w/o t. s." were computed using an architectural feature that allows us to disable toggle-suppression. Thus, the top portion of these bars represents the amount of current that our design saves by employing toggle suppression. Interestingly, the maximum power application without toggle suppression is floating-point multiply intensive, rather than load intensive. This is because without t.s., the load unit no longer needs a load instruction as an excuse to burn data memory power.

5.3 Network Power

We continue our investigation by examining the energy cost of communication over the two types of networks. Note that the graphs do not include clock energy, and that N hops implies going through N+1 routers. All transmitted sequences maximize toggle rate.

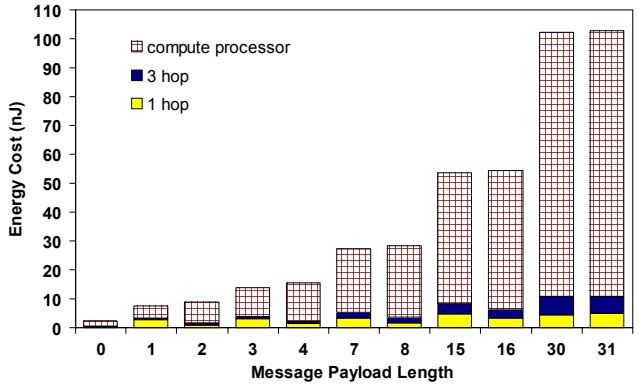

Figure 3: Energy Cost of Dynamic Network Messages

Dynamic network messages are sent as multi-word packets with a one word header. There are three major components of the energy: compute processor send/receive costs, a per-packet cost for routing, and a per-word cost. Figure 3 graphs dynamic message energy by varying the message length, and the number of hops that each message travels. As messages get longer, the overhead of route initiation is greatly amortized. Odd payloads are more power intensive because they allow perfectly inverted messages.

The static router routes single-word packets. In this case, in addition to any compute processor send/receive costs, a fixed cost (approximately 504 pJ, 90% due to instruction cache access) is paid for routing one to thirteen values through the router on a given cycle, with an amortized cost of 85 pJ per 32-bit maximal-toggle word that is routed. As shown in Figure 4, the static network is more energy efficient than the dynamic network for single-word messages because it does not have the header overhead. However, for 31-word messages, the dynamic network is almost twice as good. If message words need to be delivered in a known order, advantage turns back to the static network; dynamic network programs give up performance because they have to execute code to de-multiplex messages. On a side note, the non-compute power of the static network could be reduced by almost half if the static router were modified to fetch more than one instruction at a time.

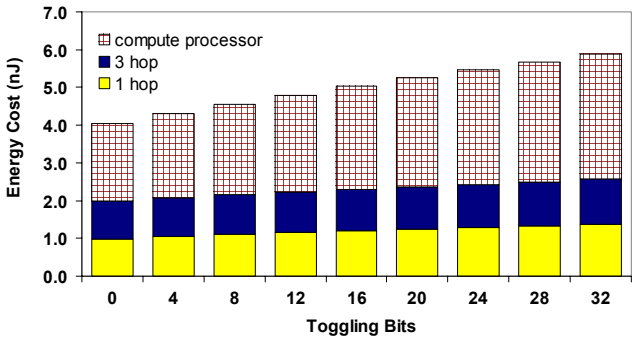

Figure 4: Energy Cost Per Word For Static Messages

The "compute processor" cost in both graphs is for move instructions to/from the register-mapped network ports on the source and receiver tiles. These could be replaced with computation instructions doing useful work (like an FPU instruction that reads from or writes to the output network port) to reduce the overhead. Realizing that each word transmitted on the network requires at least one instruction to generate it and one instruction to receive it, this data suggests that instruction energy will be larger than dynamic network transmit energy at least for mesh sizes up to 128 nodes.

5.4 Power for SHA and vpenta

To get a better sense of the power profile of the processor, we selected a pair of applications for in-depth study. We examined "SHA", an implementation of the inner kernel of the secure hash algorithm, and "vpenta", a kernel from Spec92's nasa7 benchmark. These applications were automatically parallelized using RawCC[6], the Raw compiler. These benchmarks were placed in a loop to increase the running time. SHA has little parallelism, and does not benefit significantly when running on more than four tiles. Vpenta, on the other hand, achieves 13.2x speedup on 16 tiles. Figure 5 shows the speedups versus execution on a single tile. Vpenta.small is a version of vpenta with a reduced data set, so as to reduce cache misses to a tiny fraction.

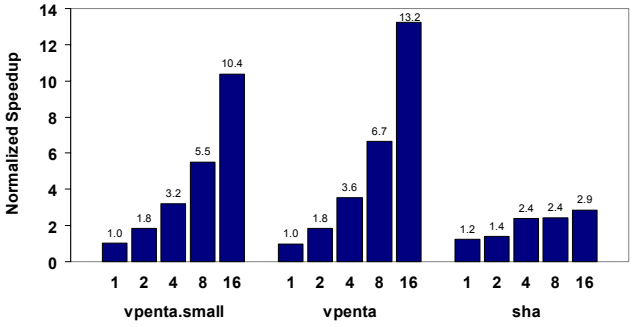

Figure 5: Speedup versus 1 tile

Unlike the previous examples, these are all actual benchmarks. We discovered that it was quite difficult to break the aggregate power into components while maintaining assurances of accuracy. Despite this, we were able to use the toggle-suppression disable mechanism to gain insight into the power consumption due to the memories. We also ran cycle-accurate simulations that counted 0-1 transitions on the networks in order to estimate the data-dependent network power, which ended up being very small. See Figure 6.

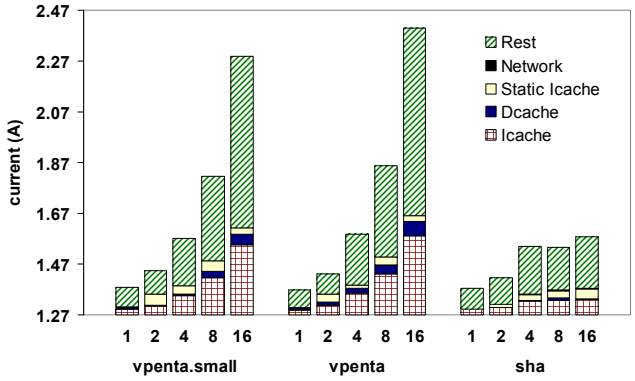

Figure 6: Average Current by Application

As Figure 6 shows, cache misses did not greatly alter power consumption for vpenta. In this graph, the current drawn is largely proportional to the amount of parallelism that the compiler is able to find. Vpenta has more parallelism, and puts the extra tiles to good use, and thus draws more current. SHA, on the other hand, beyond four tiles, draws little additional current. This is because the idle tiles stall and toggle-suppress, resulting in small power consumption.

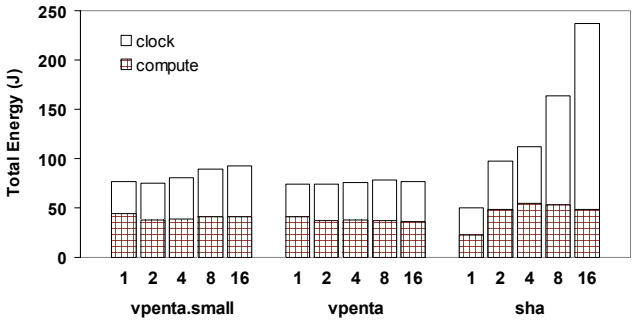

Figure 7: Energy w/ Hypothetical Tile-level Clock-gating

Adding tile-level clock gating to the processor would change this behavior, however. With such gating, the compiler chooses the number of tiles to dedicate to a program phase, so that the unused tiles can be clock-gated to save clock power. As Figure 7 shows, if clock-gating were incorporated, SHA's energy consumption would rise quickly as the number of un-clock-gated-but-idle tiles increases. Vpenta, on the other hand, makes good use of extra processing resources, with little extra energy, even with increasing clock power.

6. CONCLUSION

In this paper, we presented preliminary ideas on how power can be managed in a tiled architecture. In addition, we presented an analysis of a tiled microprocessor implementation, the Raw microprocessor, which isolates inactive components on a cycle-by-cycle basis. We found that tiling simplifies power management, even in a large chip with hundreds of functional unit clusters and memories.

[1] Michael Taylor et al. The Raw Microprocessor. *IEEE Micro*, Apr 2002.

[2] R. Ho, K. Mai, M. Horowitz. Efficient On-Chip Global Interconnects. *IEEE Symposium on VLSI Circuits*, June 2003.

[3] Michael Taylor et al. Scalar Operand Networks: On-chip Interconnect for ILP in Partitioned Architectures. *Proc. HPCA* , Feb 2003.

[4] H. Wang, et al. Power Model for Routers. *IEEE Micro*, Jan 2003.

[5] Gunther, et al. Managing the Impact of Increasing Microprocessor Power Consumption. *Intel Technology Journal*. 1Q2001.

[6] Walter Lee, et al. Space-Time Scheduling of Instruction-Level Parallelism on a Raw Machine. Proc. *ASPLOS*, 1998.