

A Modular 3D Processor for Flexible Product Design and Technology Migration

Gabriel H. Loh
 Georgia Institute of Technology
 College of Computing
 Atlanta, GA, USA
 loh@cc.gatech.edu

ABSTRACT

The current methodology used in mass-market processor design is to create a single base microarchitecture (e.g., Intel’s “Core” or AMD’s “K8”) that is used throughout all of the PC market segments from laptops to servers. To differentiate the products, manufacturers rely on speed binning, different cache sizes, and varying the number of cores. In this paper, we propose using 3D integration to provide a new, but complementary, approach to providing product differentiation. Past research on using 3D to improve performance has focused on the construction of “fully 3D” circuits where functional blocks are partitioned across two or more layers. This approach forces one of two undesirable situations: (1) all products must be implemented in, and therefore pay the cost of, 3D or (2) a 3D-implemented processor is designed for the high-end/high-performance markets and a separate 2D microarchitecture must be designed for the lower-cost markets thereby incurring significant additional design effort and engineering cost. We present a modular processor architecture where 3D can be used to enhance performance within a single unified design and also provides for a more gradual migration path toward fully 3D-integrated designs. To make this work, we describe a generic technique of using “phantom” components where the baseline processor may believe that 3D-stacked resources exist, but are currently unavailable. Simply using 3D to stack more L2 cache provides a 15.1% average performance benefit, but our proposal increases performance by 25.4%.

Categories: C.1.0 [Computer Systems Organization] Processor architectures – *General*

General Terms: Design, Performance

Keywords: 3D-integration, modular, superscalar

1. INTRODUCTION

In the past, high-volume microprocessor companies designed very different microarchitectures to target different market

segments.¹ For example, in the early 2000’s, Intel corporation had the Pentium-4 (Willamette) core for desktop, workstation and server markets, while the Pentium-M (Banias) microarchitecture targeted mobile and budget segments. Due to the skyrocketing complexity of modern general purpose processors and the related non-recurring expense (NRE) of developing new microarchitectures, companies have switched to a “converged” design methodology where a single microarchitecture design provides for all product segments. For example, Intel’s current “Next-Generation Micro-Architecture” (NGMA), also known as the “Core 2,” is the same pipeline design deployed from their low-end Celerons to their high end Xeons for the server market. Techniques such as speed binning, varying the amount of L2 cache, and varying the number of cores allows the manufacturer to retarget a single design for the different demands across market segments.

Three dimensional circuit integration is an emerging technology that has recently received much attention among computer architects [19, 20, 22, 24, 28, 37]. The technology stacks multiple layers of integrated chips to provide greater device density and shorter interconnects due to the ability to place and route in the third dimension. The reduction in wire-length throughout the processor, which today is heavily wire-dominated [4], can provide simultaneous performance and power benefits. For example, one can partition an SRAM by stacking the bitcell arrays to reduce the lengths of the wordlines which speeds up the SRAM access latency as well as reduces the power spent in charging/discharging the long wordlines [32, 38, 53].

Despite the advantages, 3D integration is not without its costs. The new manufacturing equipment and additional fabrication steps increase the cost per part. A 3D-stacked chip may also increase chip temperatures [4, 23, 37], thereby requiring more aggressive system cooling solutions that increase overall system cost. 3D integration may not be practical for all market segments, as mobile devices may not be able to support larger/more complex cooling systems and low-end computer systems using low-cost processors (e.g., Intel Celeron, AMD Sempron) may not need the additional performance and the extra cost would be intolerable in a market segment where profit margins are already very tight.

We would like the performance advantages of 3D for the markets that need it. Designing a completely-3D processor such as the 3D Pentium 4 examined by Black et al. [5]

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CF’08, May 5–7, 2008, Ischia, Italy.

Copyright 2008 ACM 978-1-60558-077-7/08/05 ...\$5.00.

¹By “microarchitecture” or “core” we mean the main execution pipeline consisting of the processor front-end, out-of-order execution engine and L1 caches. The L2 cache, memory interface controller, and other logic are considered to be outside of the core; these components are sometimes called the “uncore.”

would either force all market segments to adopt (and pay the price for) 3D integration, or the manufacturer would have to shoulder the additional NRE of designing two separate microarchitectures: one using 3D for the high-end markets, and a conventional 2D design for the remaining markets. The objectives of maintaining a single converged core design and the desire to not inflict the cost of 3D on markets that do not need it seem to be in direct conflict. In this work, we propose a new 3D microarchitecture that addresses this conflict, thereby enabling the manufacturer to use 3D as a new technique for providing product differentiation across market segments. Note that we do not propose to use 3D instead of standard practices such as changing L2 cache sizes and core counts, but our proposed 3D design approach adds an additional and orthogonal technique to the computer architect's toolbox. As 3D technology matures and the cost of implementing processors in 3D decreases, our proposed technique also provides a more gradual and lower-risk migration path away from traditional 2D designs.

This paper is organized as follows. Section 2 provides a brief overview of 3D integration and discusses related research. Section 3 details our proposal for our 3D microarchitecture. Section 4 presents a detailed performance evaluation of our proposal, and Section 5 considers the thermal impact of our stacked architecture. Section 6 draws some final conclusions and discusses future research directions.

2. 3D INTEGRATION

In this section, we first present a brief overview of three-dimensional chip-stacking technology and then briefly review related research.

2.1 The Technology

The basic idea of 3D integration is not new, but due to recent advances in the manufacturing technology, computer architects have taken a new interest in the potential for 3D-implemented microprocessors. Wafer-bonding appears to be the technology that is gaining the most traction in industry [5, 15, 30]. In copper-copper (Cu-Cu) wafer bonding, the circuits for each layer are constructed on separate wafers using conventional 2D/planar fabrication techniques. After wafer construction, additional inter-layer or die-to-die (d2d) vias are constructed and the entire stack is aligned and bonded. Without getting into the process manufacturing details, the salient characteristics of current 3D-integrated technologies are that the d2d vias are very small and very fast. At a pitch of $3\mu\text{m}$ per d2d via [8], one can achieve a density of over 100,000 d2d vias per mm^2 . The latency of sending a signal from one layer to another through the via is also very fast; well under a single gate delay [32]. From the perspective of the computer architect, the d2d vias can be thought of as a small component of conventional metal/wire delay. In this study, we assume a face-to-face Cu-Cu bonding process, which appears to be favored by Intel [4], although the techniques that we propose in this paper can be easily implemented in a face-to-back process as well.

2.2 Related Research

Many recent research efforts have explored the possibilities of splitting or partitioning a microprocessor pipeline across two (or more) layers of 3D-stacked silicon. Some studies have focused on splitting individual functional unit blocks (FUB) [25, 32–34, 36, 38, 53, 54], others have explored main-

taining 2D FUBs but re-floorplanning the design to stack FUBs on top of each other [7, 17], and others have proposed designs that completely repartition the microarchitecture [5, 37, 55]. All of the works referenced above share the common trait that the resultant designs are completely 3D and thereby force the undesired choice of either pushing 3D into markets that do not need it or incurring the expense of designing multiple microarchitectures.

The most related research involves using 3D to provide “snap-on” functionality. Mysore et al. propose to start with a standard 2D processor core, and then use 3D to stack additional profiling and debugging (“introspection”) engines [28]. The idea is that for mass-volume manufacturing, only the baseline 2D processor core will ever be produced, but the small population of software developers (esp. for operating systems), OEMs and other special users, can pay the extra cost of the stacked 3D introspection engines to greatly assist and accelerate the development and debugging of new products and platforms. Similarly, Madan and Balasubramonian propose to use 3D integration to provide a snap-on reliability engine (a la DIVA [2]) to enhance the reliability, availability and dependability for mission-critical systems [24]. These approaches effectively avoid the problem of forcing the majority of the market to shoulder the costs of 3D integration while providing benefits for a few that need it. The microarchitecture proposed in this paper has similarities in that it effectively uses 3D's snap-on attribute to provide optional functionality. There are two primary differences. At a high level, our proposal has much wider applicability as our 3D-stacked design can be deployed in the server, workstation and hardcore gaming markets (contrast this to the introspective 3D processors that only target a relatively small market [29] and the reliable 3D processors which consist of a subset of the server market). At a low level, the previous approaches primarily place entire FUBs (i.e., the optional introspection and reliability engines) on the stacked die where the logic to support an optional/missing structure is relatively straightforward. In this work, we propose FUB designs that both span layers and can continue to work when the optionally stacked top-halves of the FUBs are missing.

3. STACKABLE MICROARCHITECTURE COMPONENTS

The overall theme for our 3D-stacked microarchitecture is to design components that are split across silicon layers, and then we “fake” certain control signals when the stacked portions are not actually present. We now describe our changes for several key pipeline components.

3.1 Stacked Instruction Schedulers

In processors supporting out-of-order execution, the size of the instruction scheduler or reservation station (RS) entries often becomes a limiting factor to the amount of exposable instruction level parallelism (ILP). Adding more RS entries typically increases the processor's instructions-per-cycle (IPC) rates, but the complexity of the circuits may cause the clock speed to decrease, resulting in a net performance loss. For example, Figure 1(a) shows an 8-entry RS and Figure 1(b) shows a 16-entry RS. The achievable clock frequency for the larger RS will be decreased because the substantial increase in the bus length, as well as capacitive loading from the additional comparators, drastically

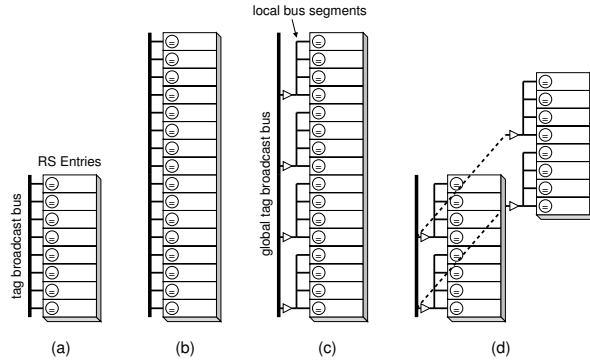


Figure 1: RS entry organizations with (a) eight RS entries, (b) 16 entries, (c) 16 entries with segmented buses, and (d) 16 entries, 3D-stacked, with segmented buses.

increases the latency of tag broadcast [31]. Figure 1(c) shows a 16-entry RS using segmented broadcast buses [21]; in this case, the loading due to the comparators is greatly reduced, but the length of the bus can still have a significant impact on clock speed. The segmentation can also save power because any segment where all RS entries are invalid need not repeat the tag broadcast. In our 3D design, we make use of this segmented approach, but the segments are vertically stacked (Figure 1(d)). This layout is unique to 3D, and the advantage is that we can reap the full benefits of tag bus segmentation, but the overall length of the bus has not increased. When the stacked layer is not present, it simply appears to the bottom layer that all entries are unused and the segment repeaters are turned-off (either power-gated or tri-stated). The (non-present) stacked layer implements a *phantom* set of RS entries that always appear to be empty with respect to the wakeup logic. Segmentation also limits the number of die-to-die vias needed since the bus can be routed between layers at a single point (the start of the segment) and then fanned out across the top layer without any additional vertical connections.

The other critical component of instruction scheduling is the select logic or *picker*. Normally, each RS entry has a set of BID and GRANT ports to the picker to ask for and receive access to functional units. When granted, the RS entry accesses its payload RAM entry which forwards the relevant information to the functional units for execution (possibly accessing the physical register file before execution depending on the microarchitecture). In our stacked scheduler design, we have twice as many RS entries, but naively increasing the number of picker ports can also have a significant impact on the clock frequency [41]. Instead, we make the stacked RS entries share a set of BID and GRANT ports. To do this, either entry can assert the BID signal through a wired-NOR pull-down on the BID port, which does not have much impact on clock speed since the loading increases by a single drain capacitor. In parallel with the picker, the two RS entries perform a small local one-of-two pick based on which entry is older. This local pick is much faster than the full picker, and has plenty of time to set up the routing on a two-to-one GRANT demux. When the global GRANT signal returns from the picker, it gets sent to one of the two entries

which can then proceed to issue. Note that this organization also allows the two RS entries to share a single read port from the payload RAM since only one of the two can possibly issue in a given cycle. The closest prior work that we are aware of is the “Reloaded” picker design of Sassone et al. where there are fewer picker ports than RS entries [41]. In their work, the ports can be dynamically allocated which requires more complex bookkeeping than our statically partitioned picker approach, but the Reloaded picker has the potential to scale to much larger picker sizes (i.e., statically sharing a single picker port between more than two RS entries will likely result in significant IPC reductions).

The allocation stage attempts to place instructions in the bottom layer first (for thermal reasons similar to those described in the Thermal Herding approach [37]). This also has the benefit that, when possible, it avoids placing instructions directly on top of each other which minimizes the effects of delaying instruction execution when both RS entries are ready but the shared picker port only allows one to execute.

When the stacked layer is not present, we can fake its presence by doing the following. First, the local BID pull-down is disabled which makes the phantom RS entry appear to always be in a not-ready state. Second, the input from the phantom entry for the local picker is set to be invalid/unoccupied so that the bottom RS entry will always be picked. Third, the bits corresponding to the allocation usage vector for the phantom layer will always be set so that the allocator believes that the stacked RS entries are always occupied and therefore will never attempt to place an instruction in any of these (non-present) entries. It is interesting to note that this approach provides different “views” of the RS at the different interfaces; by faking the signals appropriately, the allocator believes the stacked RS entries are all full while the picker believes that they are all empty. By using this concept of phantom structures, we allow the vast majority of the regular processor control logic to behave as it normally does.

3.2 Partitioning the Load and Store Queues

The reservation station entries manage the dataflow scheduling of “regular” arithmetic instructions. Load and store instructions, however, require extra handling due to the dataflow dependencies created through memory addresses. A load instruction may or may not be dependent on an earlier store instruction depending on the final address calculations of both the load and the store. The load and store queues (LDQ and STQ, respectively) are responsible for buffering these instructions and maintaining the correct order of execution with respect to these memory dependencies. Increasing the instruction capacity of these structures, however, is arguably even more difficult than building a large RS. As a result, much research has gone into devising scalable load and store scheduling structures [3, 10, 43, 45, 46, 50, 51].

We use a communication-partitioned LDQ/STQ architecture that leverages well-studied properties of load-store communication patterns. In particular, Baugh and Zilles proposed a load-store queue (LSQ) that consists of a relatively small fully-associative STQ (which they call the store forwarding buffer or STB) and a larger and more scalable, but not directly searchable, structure for the majority of loads and stores [3]. The majority of stores that are not likely to participate in forwarding get allocated to the larger LSQ

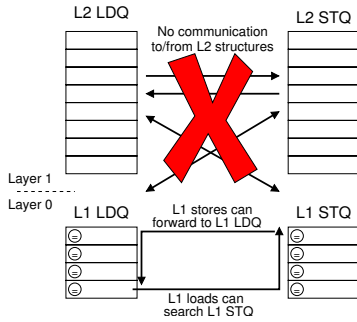


Figure 2: Organization and communication of the stacked LDQ and STQ structures.

structure. The smaller number of remaining stores reside in the STB. Our proposed stacked LSQ architecture is a generalization of the Baugh and Zilles LSQ. We start with a partitioned LDQ/STQ organization that is typical of current processor designs (as opposed to a unified LSQ). We then stack larger, direct-mapped “L2” load and store queues to the second layer (Figure 2). The L2 LDQ can only access the cache; loads allocated into this structure cannot search either of the STQs for forwarded data. Similarly, the L2 STQ only stores addresses and data for the eventual writeback when the stores commit; this structure cannot forward data to either LDQ. This severely limited functionality in the L2 structures makes it much easier to build larger LDQ/STQs. The lack of communication between the L1 and L2 LDQ/STQ structures also significantly limits the number of die-to-die vias needed to implement our LSQ architecture.

Any loads and stores that are predicted to be involved in store-to-load forwarding are placed in the fully-associative LDQ or STQ. All other loads and stores allocate into the L2 structures. The one exception to this allocation rule is that if the L2 LDQ (STQ) is full, then a load (store) can allocate into the L1 LDQ (STQ) regardless of any dependence/forwarding predictions. This is also the mechanism by which we implement the phantom versions of these queues: when the stacked structures are not present, the allocation signals are faked such that the allocator believes that the L2 LDQ and STQ are both always full.

Since the loads and stores are allocated in the load and store queues based on predictions, it is always possible that a load executes with the wrong data (e.g., it used a value from the cache when it should have received forwarded data). To guarantee correct execution, we leverage the previously proposed technique of commit-time filtered load re-execution [6, 39]. When re-execution indicates that a load should have received a forwarded value, the pipeline is flushed since the load’s forward slice has now been contaminated with a bogus value. The processor then records the PC’s for both the offending store and load in predictor tables. These predictors are very similar in structure to those used to link store-load dependencies in the Store Queue Index Prediction and Fire-and-Forget approaches [45, 51]. In particular, we use a Store PC Table (SPCT) to link stores to mis-ordered loads, a Store Forwarding Predictor (SFP) and a Load Receiving Table (LRT). The SFP and LRT are both PC-indexed tables of resetting counters. On a re-execution-induced pipeline flush or a successful store-to-load forwarding between the L1 STQ

and LDQ, we set entries in both the SFP and LRT corresponding to the store-load pair to the maximum counter value. So long as this counter is non-zero, the allocator will send these instructions to the L1 LDQ/STQ. Each time a load or store commits, was not involved in forwarding, and did not cause a re-execution pipeline flush, the processor decrements the corresponding predictor counter by one, saturating at zero.

3.3 Extensible SRAM Structures

Many important structures in a processor are composed of relatively simple SRAM arrays. The sizes, associativities, and port requirements may vary, but the underlying structures are fundamentally very similar. Examples includes caches, TLBs, branch predictors, reorder buffers, register files and others. Many previous 3D SRAM designs have proposed organizations that split the bit-arrays across two or more layers [25, 32, 38, 53]. As discussed earlier in this paper, such an approach can provide significant performance and power benefits, but it is unfortunately incompatible with our stated goal of making 3D completely optional.

An SRAM may contain n sets, where a set could be a register in a register file, an instruction entry in a reorder buffer, or a traditional “set” in a cache. We place one-half of the sets on each layer of silicon, as shown in Figure 3(a). The first layer of the row decoder tree resides on the bottom die (root node). One of the two outputs can select the row decoder sub-tree on this bottom die, and the other output gets routed to the stacked upper layer. In phantom mode, we simply need to force the input for the first row decoder level to always be zero, in which case the upper (non-existent) rows can never be selected (Figure 3(b)).

Tagged SRAM structures require an additional modification. Typically, for an n -bit address, if k bits are used to select the set and offset, then the remaining $n-k$ bits form the tag. However, the number of sets in our stacked SRAM varies with the number of layers. If the single-layered 2D case requires k bits for the set and offset, then the stacked version would require $k+1$ bits since it has twice as many sets. Instead of varying the width of the tags depending on the presence of the stacked layer, we simply use an extra tag bit in the baseline 2D case. This means that the tag match for the 3D-stacked versions may be slightly slower because one bit of the tag comparison is redundant (i.e., the bit is already uniquely specified by the set number), but the latency difference between, for example, a 25-bit and a 26-bit comparison is quite small. For untagged structures such as the reorder buffer, this is not an issue.

We make use of this optionally-stacked SRAM approach for many of the SRAM-based components in the processor. These include the L1 caches, TLBs, the physical register file/ROB, various fetch queues, prefetcher history tables, the BTBs and the return address stack. The full details are provided in Section 4.

3.4 Reconfigurable Branch Prediction

We assume a baseline processor configuration with a global-history-based gshare branch predictor [26]. For our stacked configuration, we could simply use the stacked SRAM approach to implement a larger predictor table. We instead choose a different approach that allows the predictor to effectively switch between two different prediction algorithms: gshare and TAGE [44]. The TAGE predictor consists of a

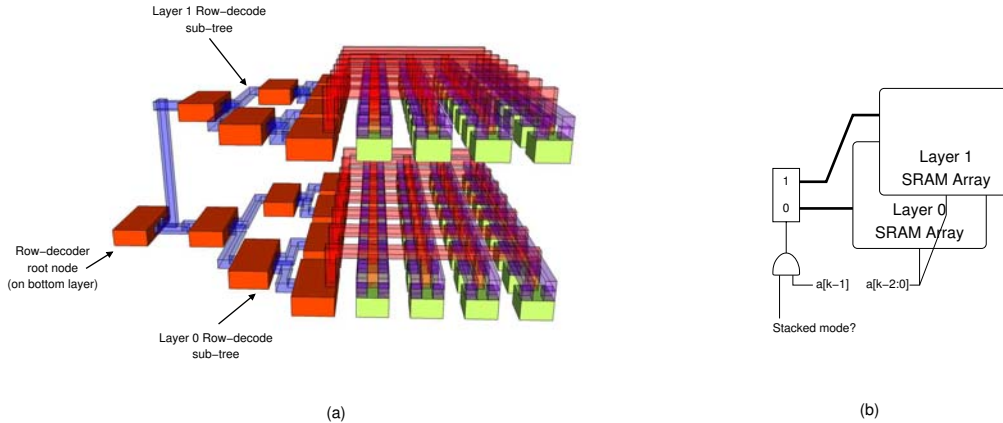


Figure 3: (a) Side view of a two-layer stacked SRAM array, and (b) a schematic view of configuring the row-decode logic for phantom mode.

series of cascaded, partially-tagged tables. Prediction occurs by performing parallel lookups in each table. The first table (an untagged, bimodal predictor [49]) provides the default prediction. A (partial) tag hit in any subsequent tables can override any of the previous predictions. Each table uses a (geometrically) longer branch history, but the organization of the tables coupled with well-tuned update rules cause the predictor to tend to use the longest history necessary to make a prediction, but no longer than that.

To support modular, stacked operation, we modify the TAGE predictor as follows. In single-layer operation, the first table acts as a gshare predictor which uses an exclusive-or of the branch address and the program counter to select a prediction. For the remaining TAGE tables, we fake their presence by simply wiring all of the HIT signals from their tag comparators to zero (indicating a miss). In this fashion, the baseline predictor effectively acts like a TAGE predictor where the overriding tables just happen to always miss, and the first table has a slightly different indexing function. In multi-layer mode, we can easily make the first predictor table revert to behave like a bimodal predictor by simply replacing the branch history in its hash computation with zeroes. This approach provides a two-algorithms-in-one branch predictor, where the more sophisticated algorithm is only employed in the stacked configuration where higher accuracy is crucial to keep the larger instruction buffers (RS, LSQ, ROB) filled with useful work.

3.5 Other Options

Our stacked processor organization provides for a larger effective instruction window for higher ILP, but we do not increase the overall *width* of the machine (fetch width, issue width, etc.). We decided that given the goal of making 3D stacking completely optional, it would be very difficult to engineer a solution where the stacked layer provides additional issue width. For example, if we want to add capacity to issue two additional instructions per layer, the RS would somehow have to be modified to support tag broadcast and wakeup for these extra instructions, the payload RAM would need additional ports, and the result bypass network would become a horrible “rat’s nest” of wires crossing back and forth between every layer’s ALUs. As a result, there remain

a few remaining blocks for which we do not extend their processing capacities. This includes the fetch unit, decode logic, renamer and allocator, the architected register file, and the commit logic.

A large number of current processors already provide multiple cores, and in the future *all* processors will likely be multi-core. Our stacked microarchitecture can be tiled in a fashion similar to existing multi-core architectures. We re-emphasize that the proposed 3D organization is not meant as a replacement for multi-core, but it provides an additional dimension for manufacturers to diversify their products while maintaining the single-microarchitecture design methodology. We will discuss multi-core options in more detail in Section 5.

4. EXPERIMENTAL EVALUATION

Having detailed the organization of our optionally stackable 3D microarchitecture, we now present the experimental validation of our proposal.

4.1 Methodology and Baseline Microarchitecture

For our performance analysis, we perform simulations using a cycle-level simulator built on top of a pre-release version of SimpleScalar for the x86 ISA [1]. Figure 4(a) shows the baseline pipeline organization implemented by our simulator, which is based off of one core of the 65nm Intel Core 2 Microarchitecture. Our simulator models many low-level details of the microarchitecture including the multiple stages of x86 decoding, limited decoding bandwidth for complex instructions that require microsequencer assistance, decomposition of x86 macro instruction to micro-op (μop) flows, insertion of additional μops to handle merging of partial register writes and REP prefixes, execution port binding [41], true speculative scheduling with decoupled broadcast-select and execute-bypass loops, wrong-path fetching past multiple branches, additional latencies between branch misprediction detection and front-end recovery, a senior store queue, atomic x86 commit (no commit until all μops have completed and are free of faults) and detailed cache architectures including all inter-level buses, MSHRs and writeback/victim

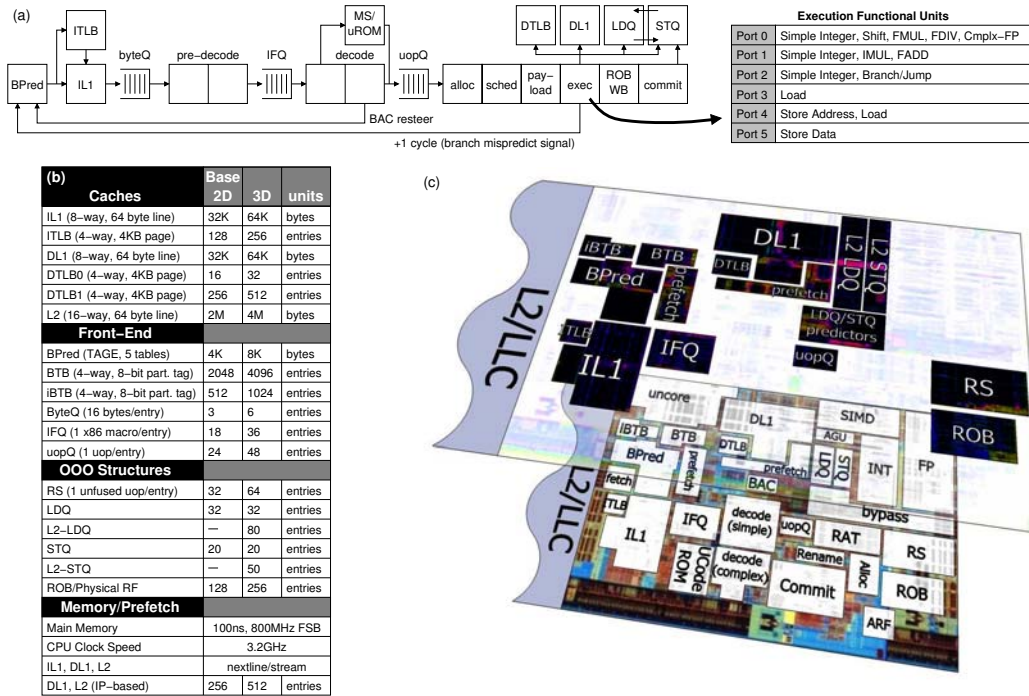


Figure 4: (a) Baseline pipeline organization with a list of functional units and port bindings, (b) detailed parameter list for 2D and 3D configurations, and (c) floorplan of baseline processor (light blocks) and the stacked layer (dark blocks).

buffers. MMX/SSE instructions and 64-bit mode instructions are not supported in this version of the x86 SimpleScalar toolset. Micro-op fusion [12] and the Core 2’s macrofusion and ESP tracker [9] are not modeled at this time. We compensate for the lack of fusion by allowing the simple decoders to decode up to two μ ops per cycle; this should not have any major effects on our results since this impacts both the baseline as well as the 3D configurations.

Figure 4(b) lists the default parameters for the baseline 2D microarchitecture as well as the 3D-stacked configuration. Both configurations can fetch 16 bytes per cycle, pre-decode four x86 macro instructions per cycle, rename/allocate four μ ops per cycle, execute up to six μ ops per cycle (assuming a mix of μ ops that matches the execution port listed in Figure 4(a)), and commit up to four μ ops per cycle. The IL1 has a latency of two cycles, the DL1 has a latency of one cycle for address computation and two cycles for the actual SRAM access for a three-cycle load-to-use latency, and the L2 has an additional nine cycle latency beyond the L1 latency. Actual cache latencies may vary due to additional queuing delays, contention for buses, etc. The stacked version also contains the additional tables/predictors to support load-re-execution and L2 LDQ/STQ allocation. The store sets bloom filter (SSBF) has 256 sets with two-way set associativity [39]. The SPCT has 2K sets, and the SFP and LRT have 1K sets each. All of these tables use 8-bit partial tags; the SFP and LRT use ten-bit counters.

Figure 4(c) also shows our baseline 2D processor floorplan and the superposition of the stacked layer with the components indicated as dark blocks. Due to the fact that

only some processor structures get extended to the stacked layers, we have some additional whitespace in the 3D-stacked floorplan. Using this extra room, we allow the L2-LDQ and L2-STQ to occupy larger footprints than their L1 counterparts. This in turn increases the capacity of each of these structures. We estimated the sizes and capacities of these structures using CACTI 4.2 [52] assuming a 65nm technology. For the remaining white space, there are several potential ways to make use of it. We list a few here, but a full evaluation of all of these ideas is well beyond the scope of this paper. Possibilities include extra analysis engines as proposed by the 3D introspective processors [28], a checker for catching soft errors [24], large amounts of decaps for increasing reliability against L_{di} problems in the power supply [14], power-gating and supply-noise controllers [27], additional/larger sleep transistors for faster response times when recharging virtual V_{da} ’s for power gating, 3D-stacked on-chip DC-to-DC voltage converters to provide multiple voltage levels [42], and ECC arrays to protect the SRAM arrays [48]. Another simple possibility is the inclusion of additional L2 cache [24]. Using a highly-banked cache design methodology such as that used for the Intel Itanium [40], one can pack extra cache capacity into the irregularly shaped regions.

We simulated integer and floating point benchmarks from the SPEC2000 and SPEC2006 benchmark suites. The results for some applications are not available due to incomplete system call support in the pre-release of the x86 version of SimpleScalar [1]. All applications make use of reference inputs. All simulations warm the caches and branch predic-

tors for 500 million instructions and then perform cycle-level simulation for 250 million instructions. We use the SimPoint 3.2 toolset to choose representative samples [16]. Note that the simulation points cover equal numbers of x86 macro instructions, which may decompose into different numbers of μ ops depending on the instruction mix per application and per simulation point. We report performance results based on the μ ops per cycle rates (μ PC). Figure 5 shows the μ PC rates for the baseline 2D processor and the average μ ops per committed x86 instruction (μ PI). All subsequent results are reported as speedups over this baseline.

4.2 Performance Results

Our 3D-stacked processor can improve performance in a variety of ways. The larger effective instruction window can potentially expose more instruction-level parallelism (ILP) and memory-level parallelism (MLP) [11], the larger caches reduce average memory access latencies, the larger branch predictor reduces pipeline flushes, and so on. We first present the overall performance results, and then dig deeper into the performance behaviors to show how these results come about.

Figure 6 shows the μ PC-rate speedups for our Modular 3D-stacked processor over the single-layer 2D baseline. The modular 3D-stacked configuration uses the second silicon layer to implement the various stacked components described earlier, and the remaining silicon area implements an additional 3MB of L2 cache, which adds up to 5MB in total. We also provide data for two additional comparisons. The STACKED L2 configuration makes use of 3D, but uses the entire available silicon area of the stacked die for L2 cache. This increases the L2 size from the baseline 2MB up to 6MB; this is similar to the stacked SRAM organization evaluated by Black et al. [4]. The second comparison is with an IDEAL configuration; this uses a unified RS without any picker/issue constrains, fully-associative LDQ and STQ where the LDQ (STQ) size is equal to the sum of the L1 and L2 LDQs (STQs) used in the stacked configuration. This provides a measure of how well our partitioned structures are working.

Performance speedups vary by application and application suite, with an overall geometric mean performance gain of 25.4%. Compared to the simpler approach of using 3D to just add more L2 cache (15.1%), our stacked architecture provides quite a bit more benefit. Our stacked approach is more balanced since some of the stacked silicon real estate does go toward additional L2 capacity, but we also use some of it to extend other processor structures. For some applications like `mcf` (CPU2000) and `art`, the additional benefit is clearly coming from the increased cache size as even the simple stacked L2 configuration delivers very large speedups. The `mcf` benchmark contains a well-known pointer chasing loop with very poor locality. The additional L2 cache capacity manages to capture a large enough portion of the working set, which reduces L2 cache misses from 43.05 misses per thousand μ ops (MPK μ) down to only 0.91 MPK μ . In the case of `art`, the performance is simply due to the fact that `art`'s memory core working set size (CWSS) is just barely larger than the size of the L2 cache for the baseline processor (Gove reports a 2.2MB CWSS, and our baseline L2 cache is only 2MB [13]). For `galgel`, the critical resource is

the DTLB where we observed a 94% reduction in the DTLB miss rate.

The increased performance of our 3D-stacked processor comes about due to a variety of reasons which vary by application. Overall, we observed larger effective instruction windows which can expose more ILP. Figure 7(a) shows the average number of μ ops allocated into various microarchitectural structures. The trends are similar across the benchmark suites, and overall we observe 71.7%, 49.0%, 51.4%, and 54.8% increases in the average number of μ ops in-flight in the RS, LDQ, STQ and ROB, respectively. Figure 7(b) shows the other major source of performance improvement which comes from reducing control and memory stalls. The larger and more accurate branch predictor decreases branch mispredictions by 29.6% and the larger L2 cache reduces misses to main memory by 35.1%. Note that the STACKED L2 configuration reduced L2 misses by even more (57.0%), but L2 misses are only one part of the overall performance equation. Attacking only the L2 certainly helps for memory-intensive applications, but there are many other workloads where the performance bottlenecks lie elsewhere.

A 25% performance improvement for a two-layer stack is significant. Previous studies that used two-die stacks to improve performance only reported benefits of about 15% [4, 55]. Recall that we took this approach to target the higher-end workstation, server and gaming markets. In these arenas, even relatively small increments in performance can translate into disproportionately larger profit margins (i.e., in the top bins, a processor that is 5% faster costs the customer much more than 5% on the final sticker price). For top-bin multi-core processors with large L2 caches, our 3D-stacked architecture can greatly improve the performance of these products, and we do so with a single unified design that does not force a 3D implementation upon the remaining market segments.

5. PERFORMANCE, POWER, THERMALS AND AREA

In this section, we take a broader view of different processor implementation styles and compare and contrast both 2D and 3D organizations with respect to performance, power, area and thermals. Our power numbers simply assume the worst-case power consumption of a dual-core processor reported by Black et al. [4]. In particular, their dual-core processor consumes 92W with 7W in their 4MB L2 cache and 42.5W per core. Therefore, a single core version with a 2MB L2 cache consumes 46W at half of the area. Besides these two 2D organizations, we also consider several 3D possibilities (Figure 8(a)). We consider the STACKED L2 organization that we evaluated earlier, a “fully” 3D implementation (SPLIT-MODE) where we assume all structures have been perfectly split between the two layers [5], a 3D dual-core implementation with cores flipped to avoid stacking hotspots (FLIP-MODE), and finally our modular 3D-stacked architecture (MODULAR). For thermal modeling, we use HotSpot version 3.1 which has support for simulating 3D-stacked integrated chips [47]. We use HotSpot's fine-grained grid model for all simulations, and Table 1 lists the modeling parameters. For the actual power distributions, we manually calibrated our power profiles to generate the same thermal distribution for the 2D dual-core processor reported by Black et al.'s work, with a worst-case hotspot of 88°C located at the

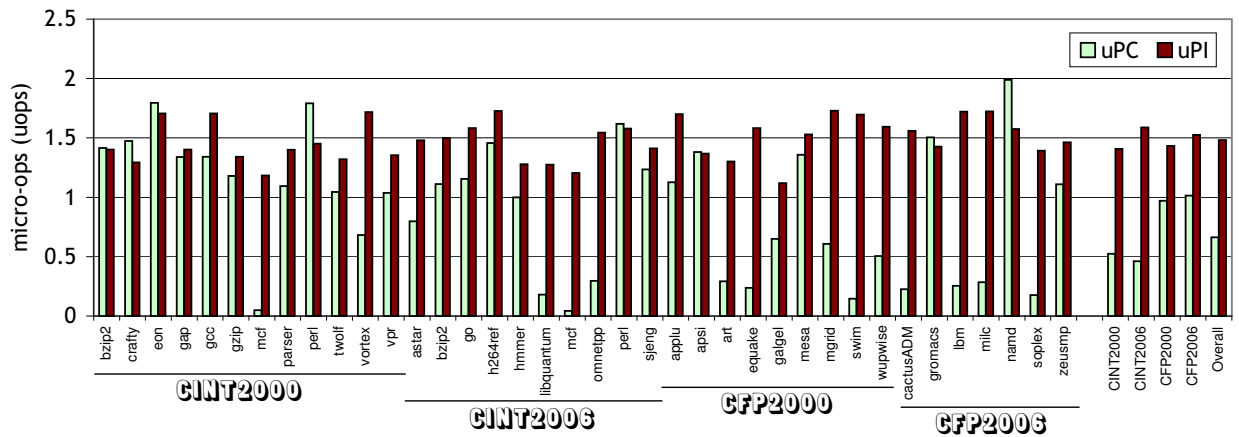


Figure 5: Baseline μ PC and μ op flow lengths for the 2D baseline processor.

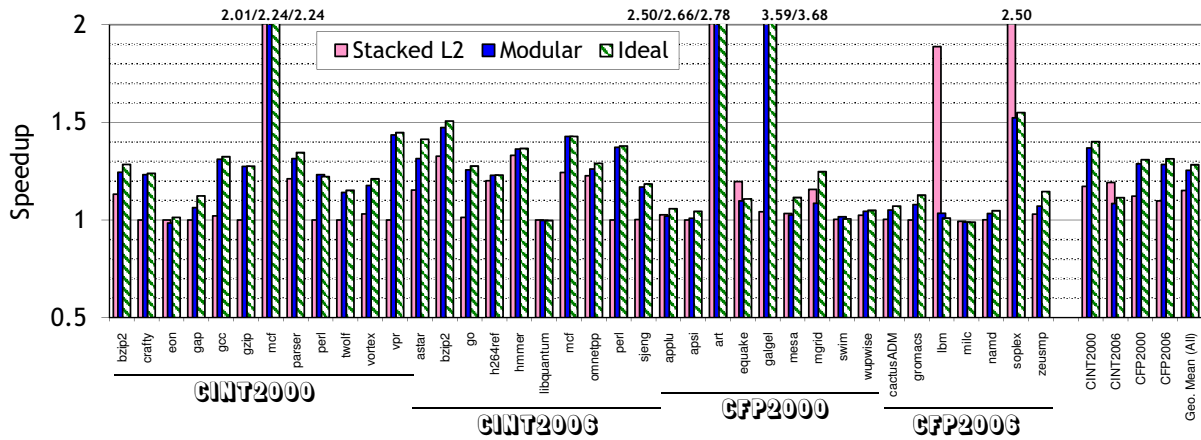


Figure 6: Performance speedup of the 3D-stacked processors.

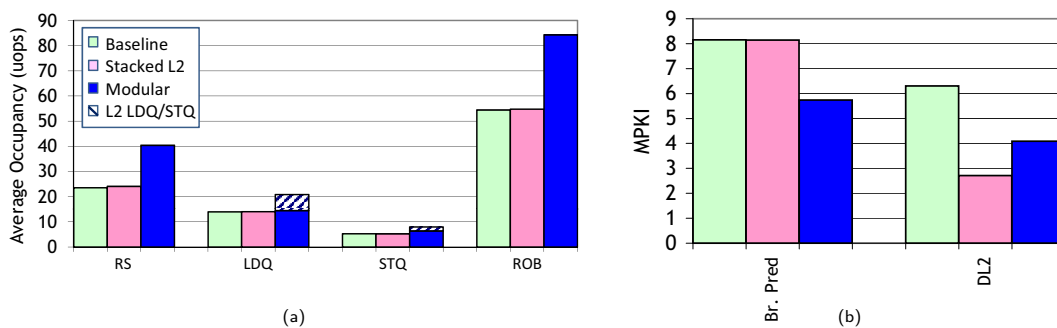


Figure 7: (a) Increasing effective instruction window size as measured by the average number of μ ops in-flight in various processor buffers, and (b) the control and memory stall reductions.

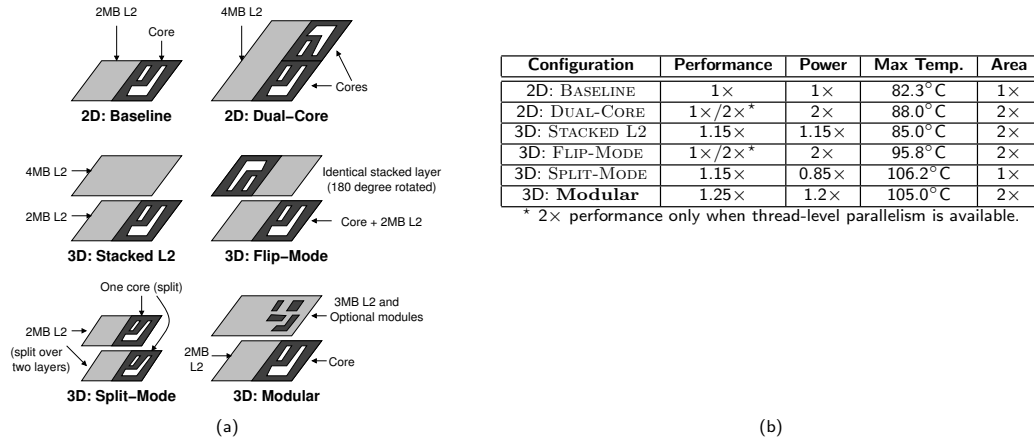


Figure 8: (a) Processor configurations considered in this section, (b) summary of tradeoffs between the different processor configurations.

Layer Thickness	
Bulk Silicon	750 μ m
Active Layer	1 μ m
Metal Layer	12 μ m
Thermal Interface Material	50.8 μ m
Face-to-face Bond	2 μ m

Thermal Resistance	
Silicon (Bulk and Active)	8.33 $\times 10^{-3}$ mK/W
Metal Layer	8.33 $\times 10^{-2}$ mK/W
Face-to-face	1.01 $\times 10^{-2}$ mK/W
Thermal Interface Material	0.2 mK/W
Ambient Temperature	40°C

Table 1: Thermal modeling parameters.

FP unit [4]. The single-core configurations have the same per-core power distribution. We assume that any additional stacked L2 cache (for STACKED L2 and MODULAR) consumes the same amount of power per unit area as the baseline 2D L2 cache. We again use CACTI 4.2 [52] to determine the power required to access the stacked structures in our MODULAR configuration (this is appropriate since these are all variants of SRAMs). For the SPLIT-MODE configuration, we assume a uniform 15% reduction in power consumption with a simultaneous 15% increase in performance as reported by Intel [4].

The different design options cover a range of tradeoffs between performance, power consumption, thermals, and total die area. Figure 8(b) summarizes the results. The thermal results are similar to those previously reported elsewhere [4, 35, 37]. Most of the options involve an increase in cost in terms of total silicon area; only the SPLIT-MODE configuration maintains approximately equal die area cost while both reducing power and increasing performance. This is not entirely surprising, since if one is willing to redesign all of the functional unit blocks in the processor to take advantage of 3D integration, one will have many more opportunities to reduce wire lengths leading to lower latencies and power consumption. The other approaches utilize designs that are primarily 2D in nature and therefore cannot fully exploit the wire-reducing attributes of 3D.

The multi-core configurations present some interesting comparisons. Using 3D to implement a dual-core processor by

simply stacking the second core in the FLIP-MODE configuration does not provide any substantial benefit. The performance, power consumption, and total silicon area are all the same, while the temperature increases. Based on these results, the only likely reason one would use 3D to stack cores in this fashion would be to reduce the overall footprint for packaging reasons. It is possible that other multi-core configurations, especially those involving asymmetric/heterogeneous cores, may find utility in a 3D-stacked organization, but a detailed evaluation of such designs is beyond the scope of this study. Another interesting observation is that the 2D DUAL-CORE configuration actually results in a higher temperature than the 3D STACKED L2 case. This is due to the fact that the worst case temperature is affected by *both* local power density as well as the total power consumption of the chip, and that stacking additional L2 does not significantly increase the power density, while adding a second core does significantly increase the total power.

The 3D MODULAR configuration delivers the largest single-threaded performance boost, but this also comes with a price. The 3D MODULAR configuration results in a similar thermal condition as 3D SPLIT-MODE, and also without the power reduction benefits. Both the MODULAR and SPLIT-MODE configurations will likely need improvements in system cooling technologies to be practical in home desktop, office workstation, and server usage scenarios. These thermal results do not account for further power reductions due to technology scaling which can play a significant role in mitigating both total power consumption and worst-case temperatures, so these thermal results may be somewhat pessimistic.

We envision two primary roles of our proposed MODULAR 3D microarchitecture. The first is a way to provide optional additional performance without significant impact on the design of the baseline 2D processors, as discussed earlier in this work. Note that this is orthogonal to using multi-core; we could stack a second layer that has two sets of the modular components (one set per core) to provide a 25% boost to each core in addition to the performance gains from any available thread-level parallelism. This may be particularly

important in light of the implications of Amdahl's Law on multi-core computing. Consider a multi-core processor with one large core for executing the sequential part of a program and N smaller cores for handling the remaining parallelizable portions of the code. Depending on how much of a program is actually parallelizable, after a certain point, adding more cores does very little to further improve performance. In this case, using the 3D-stacked layer to improve the large core's performance can have a bigger performance impact by addressing the sequential portion of the program execution.

The second role of our modular 3D design approach is to provide a smoother migration path to true 3D microarchitectures such as the 3D SPLIT-MODE configuration or the Thermal Herding microarchitecture [37] which both require 3D-splitting of individual functional unit blocks at the circuit level. Processor designers will not switch "over-night" from current 2D designs to such 3D designs. Our modular microarchitecture provides a smoother transition to 3D that helps manage the risks of adopting a new technology. While the benefits of the 3D STACKED L2 configuration are somewhat limited, we believe that the first 3D-stacked processor designs will be along these lines since the STACKED L2 configuration has relatively low redesign effort and low overall risk. Our 3D MODULAR design approach would then provide a next step to deliver more performance while continuing to mitigate the design effort and risk of 3D integration. This approach can be used for a few generations until the process technology, tools and testing support, circuit designs, and other requirements for true-3D designs have sufficiently matured so that the cost is low enough to ubiquitously use 3D from the high-performance server markets down to the low-budget commodity segments.

6. CONCLUSIONS

The engineering cost for architecting and implementing a new microarchitecture is very significant. While an interesting technology, 3D integration is not likely to be practical if it multiplies a microprocessor company's development costs by a factor of two or more (assuming completely different design efforts for each market segment). In this work, we have shown that it is possible to develop a single microarchitecture design that simultaneously supports 2D and 3D implementations, thereby providing an additional means of differentiating products across market segments.

Our proposed modular 3D organization is but one possible approach to providing a single design for a wide range of market segments. There may be other stackable microarchitectures that provide greater benefits and/or lower costs. If one considers non-traditional architectures such as the IBM Cell BE [18], one could imagine extensible implementations where optional, additional layers provide more Synergistic Processing Elements (for Cell) to enhance performance. We have shown that for traditional microarchitectures, we can successfully use 3D to enhance performance in select market segments while maintaining a single design methodology, and we believe that our modular 3D-stacked approach provides for an effective migration path toward fully 3D-integrated designs.

Acknowledgments

This project was funded by NSF grant CCF-0643500; support was also provided by the Focus Center for Circuit & System Solutions (C2S2), one of five research centers funded

under the Focus Center Research Program, a Semiconductor Research Corporation Program. Equipment was provided by a grant from Intel Corporation.

7. REFERENCES

- [1] Todd Austin, Eric Larson, and Dan Ernst. SimpleScalar: An Infrastructure for Computer System Modeling. *IEEE Micro Magazine*, pages 59–67, February 2002.
- [2] Todd M. Austin. DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design. In *Proceedings of the 32nd International Symposium on Microarchitecture*, pages 196–207, Haifa, Israel, November 1999.
- [3] Lee Baugh and Craig Zilles. Decomposing the Load-Store Queue by Function for Power Reduction and Scalability. *IBM Journal of Research and Development*, pages 287–297, March–May 2006.
- [4] Bryan Black, Murali M. Annavaram, Edward Brekelbaum, John DeVale, Lei Jiang, Gabriel H. Loh, Don McCauley, Pat Morrow, Donald W. Nelson, Daniel Pantuso, Paul Reed, Jeff Rupley, Sadas Shankar, John Paul Shen, and Clair Webb. Die-Stacking (3D) Microarchitecture. In *Proceedings of the 39th International Symposium on Microarchitecture*, Orlando, FL, December 2006.
- [5] Bryan Black, Don Nelson, Clair Webb, and Nick Samra. 3D Processing Technology and its Impact on IA32 Microprocessors. In *Proceedings of the 22nd International Conference on Computer Design*, pages 316–318, San Jose, CA, USA, October 2004.
- [6] Harold W. Cain and Mikko H. Lipasti. Memory Ordering: A Value-Based Approach. In *Proceedings of the 31st International Symposium on Computer Architecture*, pages 90–101, München, Germany, June 2004.
- [7] Jason Cong, Ashok Jagannathan, Yuchun Ma, Glenn Reinman, Jie Wei, and Yan Zhang. An Automated Design Flow for 3D Microarchitecture Evaluation. In *Proceedings of the 11th Asia South Pacific Design Automation Conference*, pages 384–389, Yokohama, Japan, January 2006.
- [8] Shamik Das, Andy Fan, Kuan-Neng Chen, and C. S. Tan. Technology, Performance, and Computer-Aided Design of Three-Dimensional Integrated Circuits. In *Proceedings of the International Symposium on Physical Design*, pages 108–115, Phoenix, AZ, USA, April 2004.
- [9] Jack Doweck. Inside Intel Core Microarchitecture and Smart Memory Access. White paper, Intel Corporation, 2006. <http://download.intel.com/technology/architecture/sma.pdf>.
- [10] Amit Gandhi, Haitham Akkary, Ravi Rajwar, Srikanth T. Srinivasan, and Konrad Lai. Scalable Load and Store Processing in Latency Tolerant Processors. In *Proceedings of the 32nd International Symposium on Computer Architecture*, pages 446–457, Madison, WI, USA, June 2005.
- [11] Andy Glew. MLP Yes! ILP No! Memory Level Parallelism, or, Why I No Longer Worry About IPC. In *Proceedings of the ASPLOS Wild and Crazy Ideas Session*, San Jose, CA, USA, October 1997.

- [12] Simcha Gochman, Ronny Ronen, Ittai Anati, Ariel Berkovitz, Tsvika Kurts, Alon Naveh, Ali Saeed, Zeev Sperber, and Robert C. Valentine. The Intel Pentium M Processor: Microarchitecture and Performance. *Intel Technology Journal*, 7(2), May 2003.
- [13] Darryl Gove. CPU2006 Working Set Size. *Computer Architecture News*, 35(1):90–96, March 2007.
- [14] Ed Grochowski, David Eysers, and Vivek Tiwari. Microarchitectural Simulation and Control of di/dt-induced Power Supply Voltage Variation. In *Proceedings of the 8th International Symposium on High Performance Computer Architecture*, pages 7–16, 2002.
- [15] K. W. Guarini, A. W. Topol, M. Jeong, R. Yu, L. Shi, M. R. Newport, D. J. Frank, D. V. Singh, G. M. Cohen, S. V. Nitta, D. C. Boyd, P. A. O’Neil, S. L. Tempest, H. B. Pogge, S. Purushothaman, and W. E. Haensch. Electrical Integrity of State-of-the-Art 0.13 μ m SOI CMOS Devices and Circuits Transferred for Three-Dimensional (3D) Integrated Circuit (IC) Fabrication. In *Proceedings of the International Electron Devices Meeting*, pages 943–945, December 2002.
- [16] Greg Hamerly, Erez Perelman, Jeremy Lau, and Brad Calder. SimPoint 3.0: Faster and More Flexible Program Analysis. In *Proceedings of the Workshop on Modeling, Benchmarking and Simulation*, Madison, WI, USA, June 2005.
- [17] Michael Healy, Mario Vittes, Mongkol Ekpanyapong, Chinnakrishnan Ballapuram, Sung Kyu Lim, Hsien-Hsin S. Lee, and Gabriel H. Loh. Multi-Objective Microarchitectural Floorplanning for 2D and 3D ICs. To appear in the *IEEE Transactions on Computer Aided Design*, 2007.
- [18] H. Peter Hofstee. Power Efficient Processor Architecture and the Cell Processor. In *Proceedings of the 11th International Symposium on High Performance Computer Architecture*, pages 258–262, San Francisco, CA, USA, February 2005.
- [19] Tae Ho Kgil, Shaun D’Souza, Ali Ghassan Saidi, Nathan Binkert, Ronald Dreslinski, Steven Reinhardt, Kristian Flautner, and Trevor Mudge. PicoServer: Using 3D Stacking Technology to Enable a Compact Energy Efficient Chip Multiprocessor. In *Proceedings of the 12th Symposium on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, USA, October 2006.
- [20] Jongman Kim, Chrysostomos Nicopoulos, Dongkook Park, Reetuparna Das, Yuan Xie, N. Vijaykrishnan, Mazin S. Yousif, and Chita R. Das. A Novel Dimensionally-Decomposed Router for On-Chip Communication in 3D Architectures. In *Proceedings of the 34th International Symposium on Computer Architecture*, San Diego, CA, USA, June 2007.
- [21] Gurhan Kucuk, Kanad Ghose, Dmitry V. Ponomarev, and Peter M. Kogge. Energy-Efficient Instruction Dispatch Buffer Design for Superscalar Processors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, Huntington Beach, CA, USA, August 2001.
- [22] Feihui Li, Chrysostomos Nicopoulos, Thomas Richardson, Yuan Xie, Vijaykrishnan Narayanan, and Mahmut Kandemir. Design and Management of 3D Chip Multiprocessors Using Network-in-Memory. In *Proceedings of the 33rd International Symposium on Computer Architecture*, pages 130–141, Boston, MA, USA, June 2006.
- [23] Gian Luca Loi, Banit Agarwal, Navin Srivastava, Sheng-Chih Lin, and Timothy Sherwood. A Thermally-Aware Performance Analysis of Vertically Integrated (3-D) Processor-Memory Hierarchy. In *Proceedings of the 43rd Design Automation Conference*, San Francisco, CA, USA, July 2006.
- [24] Niti Madan and Rajeev Balasubramonian. Leveraging 3D Technology for Improved Reliability. In *Proceedings of the 40th International Symposium on Microarchitecture*, Chicago, IL, December 2007.
- [25] John Mayega, Okan Erdogan, Paul M. Belemjian, Kuan Zhou, John F. McDonald, and Russel P. Kraft. 3D Direct Vertical Interconnect Microprocessors Test Vehicle. In *Proceedings of the ACM Great Lakes Symposium on VLSI*, pages 141–146, Washington, DC, USA, April 2003.
- [26] Scott McFarling. Combining Branch Predictors. TN 36, Compaq Computer Corporation Western Research Laboratory, June 1993.
- [27] Fayez Mohamood, Michael Healy, Sung Kyu Lim, and Hsien-Hsin S. Lee. A Floorplan-Aware Dynamic Inductive Noise Controller for Reliable Processor Design. In *Proceedings of the 39th International Symposium on Microarchitecture*, pages 3–14, Orlando, FL, December 2006.
- [28] Shashidar Mysore, Banit Agarwal, Sheng-Chih Lin, Navin Srivastava, Kaustav Banerjee, and Timothy Sherwood. Introspective 3D Chips. In *Proceedings of the 12th Symposium on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, USA, October 2006.
- [29] Shashidar Mysore, Banit Agrawal, Navin Srivastava, Sheng-Chih Lin, Kaustav Banerjee, and Timothy Sherwood. 3D-Integration for Introspection. *IEEE Micro Magazine*, 27(1):77–83, January–February 2007.
- [30] Don Nelson, Clair Webb, Don McCauley, Kartik Raol, Jeff Rupley II, John DeVale, and Bryan Black. A 3D Interconnect Methodology Applied to iA32-class Architectures for Performance Improvements through RC Mitigation. In *Proceedings of the 21st International VLSI Multilevel Interconnection Conference*, Waikoloa Beach, HI, USA, September 2004.
- [31] Subbarao Palacharla. *Complexity-Effective Superscalar Processors*. PhD thesis, University of Wisconsin, 1998.
- [32] Kiran Puttaswamy and Gabriel H. Loh. Implementing Caches in a 3D Technology for High Performance Processors. In *Proceedings of the International Conference on Computer Design*, San Jose, CA, USA, October 2005.
- [33] Kiran Puttaswamy and Gabriel H. Loh. Dynamic Instruction Schedulers in a 3-Dimensional Integration Technology. In *Proceedings of the ACM Great Lakes Symposium on VLSI*, pages 153–158, Philadelphia, PA, USA, May 2006.
- [34] Kiran Puttaswamy and Gabriel H. Loh. Implementing Register Files for High-Performance Microprocessors

- in a Die-Stacked (3D) Technology. In *Proceedings of the International Symposium on VLSI*, pages 384–389, Karlsruhe, Germany, March 2006.
- [35] Kiran Puttaswamy and Gabriel H. Loh. Thermal Analysis of a 3D Die-Stacked High-Performance Microprocessor. In *Proceedings of the ACM Great Lakes Symposium on VLSI*, pages 19–24, Philadelphia, PA, USA, May 2006.
- [36] Kiran Puttaswamy and Gabriel H. Loh. Scalability of 3D-Integrated Arithmetic Units in High-Performance Microprocessors. In *Proceedings of the 44th Design Automation Conference*, pages 622–625, 2007.
- [37] Kiran Puttaswamy and Gabriel H. Loh. Thermal Herding: Microarchitecture Techniques for Controlling HotSpots in High-Performance 3D-Integrated Processors. In *Proceedings of the 13th International Symposium on High Performance Computer Architecture*, pages 193–204, Phoenix, AZ, USA, February 2007.
- [38] Paul Reed, Gus Yeung, and Bryan Black. Design Aspects of a Microprocessor Data Cache using 3D Die Interconnect Technology. In *Proceedings of the International Conference on Integrated Circuit Design and Technology*, pages 15–18, Austin, TX, USA, May 2005.
- [39] Amir Roth. Store Vulnerability Window (SVW): A Filter and Potential Replacement for Load Re-Execution. *Journal of Instruction Level Parallelism*, 8, 2006.
- [40] Stefan Rusu, Jason Stinson, Simon Tam, Justin Leung, Harry Muljono, and Brian Cherkauer. A 1.5-GHz 130-nm Itanium 2 Processor with 6-MB On-Die L3 Cache. *IEEE Journal of Solid-State Circuits*, 38(11):1887–1895, November 2003.
- [41] Peter G. Sassone, Jeff Rupley, Edward Brekelbaum, Gabriel H. Loh, and Bryan Black. Matrix Scheduler Reloaded. In *Proceedings of the 34th International Symposium on Computer Architecture*, pages 335–346, San Diego, CA, USA, June 2007.
- [42] Gerard Schrom, Peter Hazucha, Jae-Hong Hahn, Volkan Kursun, Donald Gardner, Siva Narendra, Tanay Karnik, and Vivek De. Feasibility of Monolithic and 3D-Stacked DC-DC Converters for Microprocessors in 90nm Technology Generation. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 263–268, Newport Beach, CA, USA, August 2004.
- [43] Simha Sethumadhavan, Rajagopalan Desikan, Doug Burger, Charles R. Moore, and Stephen W. Keckler. Scalable Hardware Memory Disambiguation for High ILP Processors. In *Proceedings of the 36th International Symposium on Microarchitecture*, pages 118–127, San Diego, CA, USA, May 2003.
- [44] André Sez nec and Pierre Michaud. A Case for (Partially) TAGges GEometric History Length Branch Prediction. *Journal of Instruction Level Parallelism*, 8:1–23, 2006.
- [45] Tingting Sha, Milo M. K. Martin, and Amir Roth. Scalable Store-Load Forwarding via Store Queue Index Prediction. In *Proceedings of the 38th International Symposium on Microarchitecture*, pages 159–170, Barcelona, Spain, November 2005.
- [46] Tingting Sha, Milo M. K. Martin, and Amir Roth. NoSQ: Store-Load Forwarding without a Store Queue. In *Proceedings of the 39th International Symposium on Microarchitecture*, pages 285–296, Orlando, FL, December 2006.
- [47] Kevin Skadron, Mircea R. Stan, Wei Huang, Sivakumar Velusamy, Karthik Sankaranarayanan, and David Tarjan. Temperature-Aware Microarchitecture. In *Proceedings of the 30th International Symposium on Computer Architecture*, pages 2–13, San Diego, CA, USA, May 2003.
- [48] T. J. Slegel, R. M. Averill III, M. A. Check, B. C. Giamei, B. W. Krumm, C. A. Krygowski, W. H. Li, J. S. Liptay, J. D. MacDougall, T. J. McPherson, J. A. Navarro, E. M. Schwarz, K. Shum, and C. F. Webb. IBM’s S/390 G5 Microprocessor Design. *IEEE Micro Magazine*, 19(2):12–23, Mar/Apr 1999.
- [49] Jim E. Smith. A Study of Branch Prediction Strategies. In *Proceedings of the 8th International Symposium on Computer Architecture*, pages 135–148, Minneapolis, MN, USA, May 1981.
- [50] Sam S. Stone, Kevin M. Woley, and Matthew I. Frank. Address-Indexed Memory Disambiguation and Store-to-Load Forwarding. In *Proceedings of the 37th International Symposium on Microarchitecture*, pages 171–182, Barcelona, Spain, November 2005.
- [51] Samantika Subramaniam and Gabriel H. Loh. Fire-and-Forget: Load/Store Scheduling with No Store Queue At All. In *Proceedings of the 39th International Symposium on Microarchitecture*, pages 273–284, Orlando, FL, December 2006.
- [52] David Tarjan, Shyamkumar Thoziyoor, and Norman P. Jouppi. CACTI 4.0. Technical Report HPL-2006-86, HP Laboratories Palo Alto, June 2006.
- [53] Yuh-Fang Tsai, Yuan Xie, Narayanan Vijaykrishnan, and Mary Jane Irwin. Three-Dimensional Cache Design Using 3DCacti. In *Proceedings of the International Conference on Computer Design*, San Jose, CA, USA, October 2005.
- [54] Balaji Vaidyanathan, Wei-Lun Hung, Feng Wang, Yuan Xie, Vijaykrishnan Narayanan, and Mary Jane Irwin. Architecting Microprocessor Components in 3D Design Space. In *Proceedings of the IEEE Symposium on VLSI Design*, Bangalore, India, January 2007.
- [55] Yuan Xie, Gabriel H. Loh, Bryan Black, and Kerry Bernstein. Design Space Exploration for 3D Architecture. *ACM Journal of Emerging Technologies in Computer Systems*, 2(2):65–103, April 2006.