Journal of Instruction-Level Parallelism 9 (2007) 1-27 Submitted 2/07; published 6/07

Understanding the Impact of 3D Stacked Layouts on ILP

Manu Awasthi MANUA@CS.UTAH.EDU
Vivek Venkatesan VVENKATE@CS.UTAH.EDU
Rajeev Balasubramonian RAJEEV@CS.UTAH.EDU
School of Computing, University of Utah,
50 S. Central Campus Dr., Room 3190
Salt Lake City, UT 84112, USA

Abstract

3D die-stacked chips can alleviate the penalties imposed by long wires within microprocessor circuits. Many recent studies have attempted to partition each microprocessor structure across three dimensions to reduce their access times. In this paper, we implement each microprocessor structure on a single 2D die and leverage 3D to reduce the lengths of wires that communicate data between microprocessor structures within a single core. We begin with a criticality analysis of inter-structure wire delays and show that for most traditional simple superscalar cores, 2D floorplans are already very efficient at minimizing critical wire delays. For an aggressive wire-constrained clustered superscalar architecture, an exploration of the design space reveals that 3D can yield higher benefit. However, this benefit may be negated by the higher power density and temperature entailed by 3D integration. Overall, we report a negative result and argue against leveraging 3D for higher ILP.

Keywords: 3D die-stacked chips, on-chip wire delays, microarchitecture loops, floorplanning, performance and temperature, clustered/tiled/partitioned architectures.

1. Introduction

The vertical stacking of dies allows microprocessor circuits to be implemented across three dimensions. This allows a reduction in distances that signals must travel. Interconnects in future technologies are known to be a major bottleneck for performance and power. ITRS projections show that global and intermediate wires can incur delays of tens of cycles [1]. Interconnects can also be responsible for 50% of total chip dynamic power [2]. By reducing overall wire lengths, 3D implementations can help alleviate the performance and power overheads of on-chip wiring. The primary disadvantage of 3D chips is that they cause an increase in power densities and on-chip temperatures.

Many recent advances have been made in fabricating 3D chips (see [3] for a good overview). This technology can incur a non-trivial cost because of increased design effort, reduced manufacturing yield, and higher cooling capacities. Even though the technology is not yet mature, early stage architecture results are required to understand its potential. There are likely three primary avenues where 3D can provide benefits:

- 3D stacking of DRAM chips upon large-scale CMPs: Inter-die vias can take advantage of the entire die surface area to implement a high bandwidth link to DRAM, thereby addressing a key bottleneck in CMPs that incorporate nearly a hundred cores [4, 5].
- "Snap-on" analysis engines: Chips employed by application developers can be fitted with additional stacked dies that contain units to monitor hardware activity and

aid in debugging [6]. Chips employed by application users will not incorporate such functionality, thereby lowering the cost for these systems.

- Improvement in CPU performance/power: The components of a CPU (cores/cache banks, pipeline stages, individual circuits) can be implemented across multiple dies. By lowering the penalties imposed by long wires, performance and power improvements are possible.

The third approach above can itself be classified in two ways:

- *Folding*: a relatively small circuit that represents a pipeline stage (say, a register file) can be partitioned across multiple dies, thereby reducing the access time and energy per access to that structure.
- *Stacking*: the 2D implementation of each individual small circuit block (pipeline stage) is preserved and 3D is leveraged to stack different circuit blocks in the vertical dimension. Such an organization helps reduce communication latencies between pipeline stages.

Most recent work has focused on the *folding* approach [7, 8, 9, 10, 11, 12]. Results have shown that this can typically help reduce the delays within a pipeline stage by about 10%, which in turn can contribute to either clock speed improvements or ILP improvements (by supporting larger structures at a target cycle time). The disadvantage with the folding approach is that potential hotspots (*e.g.*, the register file) are partitioned and placed vertically, further exacerbating the temperature problem. Much design effort will also be invested in translating well-established 2D circuits into 3D. Research efforts are on-going to help realize the potential of folded 3D circuits. This paper focuses on the alternative *stacking* approach. The primary advantage of this approach is the ability to reduce operating temperature by surrounding hotspots with relatively cool structures. It also entails less design complexity as traditional 2D circuits can be re-used. A third advantage is a reduction in wire delay/power for interconnects between various microarchitectural structures.

The stacking approach has received relatively less attention in recent years. A study by Loi et al. [13] evaluates an architecture where cache and DRAM are stacked upon a planar CPU implementation. Li et al. [14] quantify the effect of the 3D stacking approach on a chip multiprocessor and thread-level parallelism. This work focuses on the effect of 3D stacking on a single core and instruction-level parallelism. A recent paper by Black et al. [15] evaluates stacking for a Pentium4 processor implementation. However, that evaluation does not provide the details necessary to understand if the stated approach is profitable for other processor models. Our work attempts to address that gap. We integrate many varied aspects (loop analysis, pipeline optimizations, SMT, automated floorplanning, distributed caches) in determining the impact of 3D on single core performance. We also provide the first evaluation of a 3D clustered architecture.

To understand the potential benefit of the stacking approach, it is necessary that we first quantify the performance impact of wire delays between microarchitectural structures. Section 2 qualitatively describes the relationships between wire delays and critical microarchitectural loops. Section 3 quantifies these relationships for single and multi-threaded superscalar cores. This data is then fed into floorplanning algorithms to derive layouts for 2D and 3D chips that optimize a combination of metrics. We show that 2D layouts are able to minimize the impact of critical wire delays. This result is more optimistic about 2D layouts than some prior work in the area (explained in Sections 2 and 3). Hence, there

is little room for improvement with a 3D implementation for traditional simple superscalar cores. However, high-ILP *clustered* architectures have a sufficiently high number of resources that not all critical structures can be co-located. Hence, we explore the design space for 3D implementations of a dynamically scheduled clustered architecture in Section 4. The performance and temperature characteristics of various clustered layouts are evaluated in Section 5 (building upon the initial work in [16]). Finally, we summarize our contributions and conclusions in Section 6.

2. Modeling Inter-Unit Wire Delays

An out-of-order superscalar processor has a number of communication paths between microarchitectural structures. For a large enough processor operating at a high frequency, some of these paths may incur multi-cycle delays. For example, the Pentium4 has a few pipeline stages dedicated for wire delay [17]. A state-of-the-art floorplanning algorithm must attempt to place microarchitectural blocks in a manner that minimizes delays for inter-block communication paths, but even the best algorithms cannot completely avoid these delays. As examples, consider the following wire delays that are encountered between pipeline stages in the Pentium4. The floating-point, integer, and load/store units cannot all be co-located – this causes the load-to-use latency for floating-point operands to be higher than that for integer operands. A recent paper by Black et al. [15] indicates that multi-cycle wire delays are encountered between the extreme ends of the L1 data cache and integer execution units. Similarly, the paper mentions that wire delays are introduced between the FP register file and FP execution units because the SIMD unit is placed closest to the FP register file. By introducing a third dimension, we can help reduce on-chip distances and the overall performance penalty of inter-block wire delays. To understand this benefit of 3D, we must first quantify the impact of inter-block wire delays on performance and evaluate if 2D floorplanning algorithms yield processors that incur significant IPC penalties from wire delays. In addition to serving as the foundation for our 3D layout study, the data produced here can serve as useful inputs for groups researching state-of-the-art floorplanning tools. It should be noted that while similar analyses exist in the literature, a few papers report inaccurate results because of simplified models for the pipeline; hence, to also enable others to reproduce our results, this section explains our loop analysis methodology in detail.

The paper by Black et al. [15] characterizes the performance and power effect of 3D on an Intel Pentium4 implementation. In that work too, 3D is primarily exploited to reduce delays between microarchitectural structures (pipeline stages). Wire delay reduction in two parts of the pipeline contribute 3% and 4% IPC improvements and many other stages contribute improvements of about 1%. The combined effect is a 15% increase in IPC in moving to 3D. While that data serves as an excellent reference point, it is specific to the Pentium4 pipeline and the cause for performance improvement in each stage is not identified. The experiments in Sections 2 and 3 help fill in the gaps and provide more insight on the performance improvements possible by eliminating intra-core wire delays. We later show that our results are less optimistic about the potential of 3D than the Intel study. We believe that specific features in the Pentium4 may have contributed to greater improvements from 3D and these improvements are perhaps not indicative of the improvements we can expect from other processors.

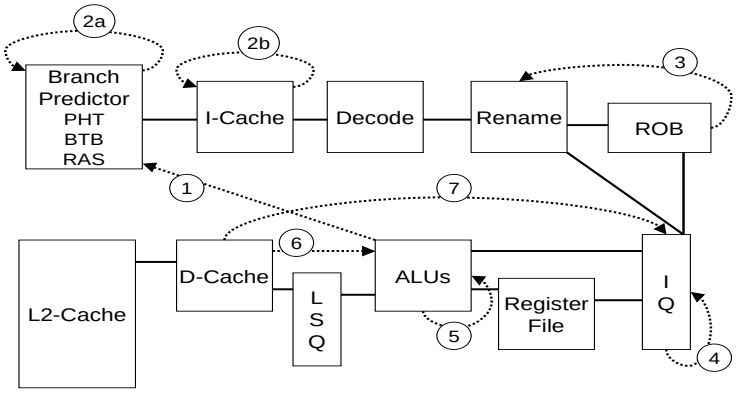


Figure 1: Critical microarchitecture loops in an out-of-order superscalar pipeline.

2.1. Critical Microarchitecture Loops

Consider the superscalar out-of-order pipeline shown in Figure 1. The pipeline is decomposed into the standard microarchitectural blocks and key data transfers between blocks are indicated with solid lines. Borch et al. [18] define a microarchitectural loop as the communication of a pipeline stage's result to the input of that same pipeline stage or an earlier stage. Loops typically indicate control, data, or structural dependences. The length of the loop is the number of pipeline stages between the destination and origin of the feedback signal. If the length of the loop is increased, it takes longer to resolve the corresponding dependence, thereby increasing the gap between dependent instructions and lowering performance. If a floorplanning tool places two microarchitectural structures far apart and introduces wire delays between them (in the form of additional pipeline stages for signal transmission), the lengths of multiple loops may be increased. Hence, to understand the IPC impact of wire delays, we must understand how the length of a loop impacts IPC. Similar, but less detailed evaluations have also been carried out in prior work (such as [18, 19, 20, 21]). The dashed lines in Figure 1 represent important loops within an out-of-order superscalar processor and each loop is discussed next.

Instruction Fetch Loop

In a simple pipeline, the process of instruction fetch involves the following steps: the PC indexes into the branch predictor system to produce the next-PC, the corresponding line is fetched from the I-cache, instructions are decoded, and when the next control instruction is encountered, it is fed back to the branch predictor system. This represents a rather large loop with a few stall cycles in fetch every time a control instruction is encountered. Introducing wire delays between the branch predictor, I-cache, and decode can severely degrade performance and this pessimistic model was assumed in HotFloorplan [20]. However, it is fairly straightforward to decouple the branch predictor and I-cache so that we instead have two short loops (labeled 2a and 2b in Figure 1). Such decoupled pipelines have been proposed by academic [22] and industrial groups [23].

In one possible implementation, the output of the branch predictor (the start of the next basic block) is fed as input back to the branch predictor. As a result, the branch

predictor system is now indexed with the PC that starts the basic block, not the PC that terminates the basic block. This allows the branch predictor to produce basic block start PCs independent of the rest of the front-end. Our results show that this change in the branch predictor algorithm has a minimal impact on its accuracy. The front-end pipeline now consists of two major tight loops: the branch predictor loop (2a) and the I-cache loop (2b). The front-end is also part of the branch mis-predict resolution loop (1), which feeds from the ALU stage all the way back to the front-end. Thus, the primary impact of introducing a wire delay between front-end pipeline stages is an increase in branch mispredict penalty. Our relative results will hold true even if a different front-end pipeline implementation (such as the next-line-and-set predictor in the Alpha 21264 I-cache [23]) is adopted, as long as the critical loops are short. Prior studies [24, 25, 20] have over-stated the IPC impact of this wire delay because the branch predictor and I-cache were assumed to not be de-coupled.

Rename Loops

The introduction of wire delays either between the decode and rename stages or between the rename and issue queue stages lengthens the penalty for a branch mispredict (loop 1). Since registers are allocated during rename, wire delays between the rename stage and the issue queue increase the duration that a register entry remains allocated (loop 3). This increases the pressure on the register file and leads to smaller in-flight instruction windows.

Wakeup and Bypass Loops

There is a common mis-conception that wire delays between the issue queue and ALUs lead to stall cycles between dependent instructions [24, 20]. This is not true because the pipeline can be easily decomposed into two tight loops – one for wakeup (loop 4) and one for bypass (loop 5). When an instruction is selected for issue in cycle N, it first fetches operands from the register file, potentially traverses long wires, and then reaches the ALU. Because of these delays, the instruction may not begin execution at the ALU until the start of cycle $N + D$. If the ALU operation takes a single cycle, the result is bypassed to the inputs of the ALU so that a dependent instruction can execute on that ALU as early as the start of cycle $N + D + 1$. For this to happen, the dependent instruction must leave the issue queue in cycle $N + 1$. Therefore, as soon as the first instruction leaves the issue queue, its output register tag is broadcast to the issue queue so that dependents can leave the issue queue in the next cycle. Thus, operations within the issue queue must only be aware of the ALU latency, and not the time it takes for the instruction to reach the ALU (delay D). The gap between dependent instructions is therefore not determined by delay D, but by the time taken for the wakeup loop and by the time taken for the bypass loop (both of these loops were assumed to be 1 cycle in the above example). The introduction of wire delays between the issue queue and ALU because of floorplanning will not impact either of these loops.

However, wire delays between the issue queue and ALU will impact another critical loop that (to the best of our knowledge) has been dis-regarded by every floorplanning tool to date. This is the load hit speculation loop (loop 7 in Figure 1). The issue queue schedules dependent instructions based on the expected latency of the producing instruction. In modern processors, such as the Pentium4 [17], the issue queue optimistically assumes

that the load will hit in the L1 data cache and accordingly schedules dependents. If the load latency is any more than this minimum latency, dependent instructions that have already left the issue queue are squashed and subsequently re-played. To facilitate this re-play, instructions can be kept in the issue queue until the load latency is known. Thus, load-hit speculation negatively impacts performance in two ways: (i) re-played instructions contend twice for resources, (ii) issue queue occupancy increases, thereby supporting a smaller instruction window, on average. If any wire delays are introduced in the pipeline between the issue queue and ALU, or between the ALU and data cache, it takes longer to determine if a load is a hit or a miss. Correspondingly, the penalties for correct and incorrect load-hit speculations increase. We also model the Tornado effect [26], where an entire chain of instructions dependent on the load are issued, squashed, and re-played on a load miss.

Delays between the issue queue and ALUs also impact branch mispredict penalty and register occupancy. They also increase the L1 miss penalty as it takes longer to re-start the pipeline after an L1 miss.

Bypassing Loops Between Groups of Functional Units

For this discussion, we assume that the functional units are organized as three clusters: integer ALUs, floating-point ALUs, and memory unit. The memory unit is composed of the load-store queue (LSQ) and L1 data cache. Bypassing within a cluster does not cost additional cycles. If wire delays are introduced between the integer and floating-point clusters, performance will be impacted for those integer operations that are data-dependent on a floating-point result, and vice versa. The introduction of wire delays between the integer cluster and memory unit impacts the load-to-use latency (loop 6 in Figure 1) and the penalties for load-hit speculation. If a single cycle delay is introduced between the memory and integer (FP) units, the load-to-use latency increases by two (one) cycles. Similarly, wire delays between levels of the cache hierarchy will increase the cache miss penalties. Table 1 summarizes the different ways that wire delays can impact performance.

Pipeline stages involved in wire delay	**Critical loops affected**
Branch predictor and L1I-Cache	Branch mispredict penalty
I-Cache and Decode	Branch mispredict penalty, penalty to detect control instruction
Decode and Rename	Branch mispredict penalty
Rename and Issue queue	Branch mispredict penalty and register occupancy
Issue queue and ALUs	Branch mispredict penalty, register occupancy, L1 miss penalty, load-hit speculation penalty
Integer ALU and L1D-Cache	load-to-use latency, L1 miss penalty, load-hit speculation penalty
FP ALU and L1D-Cache	load-to-use latency for floating-point operations
Integer ALU and FP ALU	dependences between integer and FP operations
L1 caches and L2 cache	L1 miss penalty

Table 1: Effect of wire delays on critical loops.

2.2. Floorplanning Algorithms

Floorplanning algorithms [19, 27, 28, 20] typically employ a simulated annealing process to evaluate a wide range of candidate floorplans. The objective functions for these algorithms attempt to minimize some combination of silicon/metal area, wire power, and chip temperature. In modern microprocessors, since delays across global wires can exceed a single cycle, a floorplanning tool must also consider the performance impact of introducing multi-cycle wire delays between two communicating microarchitectural blocks. The objective function in a state-of-the-art floorplanner can be represented as follows [19, 27, 28, 20] :

$$\lambda_A \times Area\_metric + \lambda_T \times Temperature + \sum_{ij} \lambda_W \times W_{ij} \times Activity_{ij} + \sum_{ij} \lambda_I \times d_{ij} \times IPC\_penalty_{ij}$$

In the equation above, $\lambda_A, \lambda_T, \lambda_W$, and λ_I represent constants that tune the relative importance of each metric (area, temperature, wire power, and IPC), W_{ij} represents the metal area (length $\times$ number of wires) between microarchitectural blocks i and j, $Activity_{ij}$ captures the switching activity for the wires between blocks i and j, the metric d_{ij} represents the distance between blocks i and j in terms of cycles, while $IPC\_penalty_{ij}$ is the performance penalty when a single cycle delay is introduced between blocks i and j. The metrics W_{ij}, d_{ij}, $Temperature$, and $Area\_metric$ are computed for every floorplan being considered, while metrics $Activity_{ij}$ and $IPC\_penalty_{ij}$ are computed once with an architectural simulator and fed as inputs to the floorplanner. The design of efficient floorplanners remains an open problem and many variations to the above objective function can be found in the literature. This work does not propose a novel floorplanning algorithm or objective function. We are using an existing floorplanning algorithm to automate the task of finding a layout that minimizes critical wire lengths. In the next section, we accurately characterize the $IPC\_penalty$ term with detailed models for the critical loops just described. The floorplanning tool takes in this input to derive 2D and 3D floorplans that (among other things) reduce the IPC penalty of wire delays.

3. The Impact of Wire Delays on Floorplanning Algorithms

3.1. Methodology

The simulator used in this study is based on Simplescalar-3.0 [29], a cycle-accurate simulator for the Alpha AXP architecture. It is extended to not only model multiple threads and separate issue queues, register files, and reorder buffer, but also the microarchitectural loops and features discussed in Section 2.1.. The single-thread benchmark suite includes 23 SPEC-2k programs, executed for 100 million instruction windows identified by the Simpoint tool [30]. The processor parameters for the base configuration are listed in Table 2. We also repeat our experiments for a core that supports the execution of two threads in SMT fashion. The SMT core has the same parameters as the single-thread processor described in Table 2, except that register file and ROB resources are doubled. Our SMT model employs the ICOUNT [31] fetch policy and all resources (except the ROB) are dynamically shared by the two threads. For the multi-threaded workload, we form a benchmark set that consists of 10 different pairs of programs. Programs are paired to generate a good mix of high IPC, low IPC, FP, and Integer workloads. Table 3 shows our benchmark pairs. Multithreaded workloads are executed until the first thread commits 100 million instructions.

Fetch queue size	16	Branch predictor	comb. of bimodal and 2-level
Bimodal predictor size	16K	Level 1 predictor	16K entries, history 12
Level 2 predictor	16K entries	BTB size	16K sets, 2-way
Branch mispredict penalty	at least 10 cycles	Fetch width	4
Dispatch width	4	Commit width	4
Issue queue size	20 Int, 20 FP	Register file size	80 (Int and FP, each)
Integer ALUs/mult-div	4/2	FP ALUs/mult-div	2/1
L1 I-cache	32KB 2-way	Memory latency	300 cycles for the first block
L1 D-cache	32KB 2-way 2-cycle	L2 unified cache	2MB 8-way, 30 cycles
ROB/LSQ size	80/40	I and D TLB	128 entries, 8KB page size

Table 2: Simplescalar simulator parameters.

Benchmark Set	**Set #**	**IPC Pairing**	**Benchmark Set**	**Set #**	**IPC Pairing**
art-applu	1	FP/FP/Low/High	bzip-fma3d	2	Int/FP/Low/High
bzip-vortex	3	Int/Int/Low/Low	eon-art	4	Int/FP/High/Low
eon-vpr	5	Int/Int/High/High	gzip-mgrid	6	Int/FP/Low/Low
mesa-equake	7	FP/FP/High/High	swim-lucas	8	FP/FP/Low/Low
twolf-equake	9	Int/FP/High/High	vpr-gzip	10	Int/Int/High/Low

Table 3: Benchmark pairs for the multi-threaded workload.

The HotFloorplan [20] tool from Virginia is used to generate 2D floorplans. For each floorplan, the tool is allowed to move/rotate blocks and vary their aspect ratios, while attempting to minimize the objective function. We also extended the tool to generate 3D floorplans with a two-phase approach similar to that described in [32]. The floorplan is represented as a series of *units/operands* (blocks in the floorplan) and *cuts/operators* (relative arrangement of blocks), referred to as a Normalized Polish Expression (NPE) [33]. Wong et al. [33] prove that a floorplan with n basic blocks can be represented as a unique NPE of size $2n - 1$. The design space can be explored by applying the following three operations. As long as the balloting property [33] holds, these operations will result in a valid floorplan: (i) swap adjacent operands, (ii) change the relative arrangement of blocks (i.e., complement the operators in NPE), and (iii) swap adjacent operator and operand. This is repeatedly performed as part of a simulated annealing process until a satisfactory value for the cost function is obtained.

For the 3D floorplan, the above basic algorithm is extended with additional moves proposed by Hung et al. [32] and is implemented as a two-phase algorithm. In the first phase, two *move* functions are introduced in addition to the three described in [33] – interlayer move (move a block from one die to another) and interlayer swap (swap two blocks between dies) – while still maintaining NPEs and satisfying the balloting property. The purpose of the first phase is two-fold: (i) minimize the area footprint ($area_{tot}$) of both the layers and the difference in the areas of each layer ($area_{diff}$), and (ii) move the delay-sensitive blocks between layers to reduce wire delays between them. The cost function used for this phase is (the equation parameters are clarified in Table 4):

$$cost_{phaseI} = \alpha_A \times area_{tot} + \alpha_{wl} \times \sum_i l_i.w_i + \alpha_d \times area_{diff}$$

Parameter	Description	Associated Weight	Value of Weight
$area_{tot}$	Total area of both dies	α_{tot}	0.05
$area_{diff}$	Area difference between dies	α_{diff}	4e5
$\sum_i l_i.w_i$	Total Wire length/delay l_i - length of wire i w_i - number of bits being transferred on wire i	α_{wl}	0.4
$\sum_{i,j} A_{olap}(i,j) \times (pd_i + pd_j)$	Power density of overlapping units $A_{olap}(i,j)$ - overlapping area between units i and j, pd_i - power density of unit i	α_{vh}	0.5
$\sum_{i_1,i_2} sharedlen(i_1,i_2) \times (pd_{i_1} + pd_{i_2})$	Lateral heat dissipation factor $sharedlen(i_1,i_2)$ - shared length between units i_1 and i_2	α_{lh}	$5e-5$

Table 4: 3D floorplanner cost function parameters.

Bulk Si Thickness die1(next to heatsink)	750μm
Bulk Si Thickness die2 (stacked die)	20μm
Active Layer Thickness	1μm
Cu Metal Layer Thickness	12μm
D2D via Thickness	5μm
Si Resistivity	0.01 $(mK)/W$
Cu Resistivity	0.0833$(mK)/W$
D2D via Resistivity (accounts for air cavities and die to die interconnect density)	0.0166 $(mK)/W$
HotSpot Grid Resolution	50x50
Ambient temperature	45 ℃

Table 5: Thermal Model Parameters.

The first phase results in two die floorplans having similar dimensions that serve as inputs to the second phase. In the second phase, no inter-die moves or swaps are allowed. This phase tries to minimize (i) lateral heat dissipation among units, (ii) total power density of all pairs of overlapping units, (iii) wire delays among units, and (iv) total area of each die using the three basic within-die moves as described in [33]. The cost function used for this stage is:

$$cost_{phaseII} = \alpha_A \times area_{tot} + \alpha_{wl} \times \sum_i l_i.w_i + \alpha_d \times area_{diff} + \alpha_{vh} \times \sum_{i,j} A_{olap}(i,j) \times (pd_i + pd_j)$$

$$+\alpha_{lh} \times \sum_{i_1,i_2} sharedlen(i_1,i_2) \times (pd_{i_1} + pd_{i_2})$$

At the end of the second phase, we obtain floorplans for two layers with favorable thermal and wire-delay properties. Finally, the L2 is wrapped around the two dies in a proportion that equalizes their area.

The average power values for each microarchitectural block are derived from the Wattch power model [34] for 90 nm technology and this is used by HotFloorplan to estimate temperatures within each candidate floorplan. Wattch's default leakage model is employed,

where a certain fraction of a structure's peak power is dissipated in every idle cycle. The leakage value is not a function of the operating temperature, thus under-estimating the power consumed by hot units. As we later show, even with this advantage, the hotter 3D architectures are unable to significantly out-perform the cooler 2D architectures. Since we are preserving the 2D implementation for each circuit block and not folding them across multiple dies, Wattch's default power models for each block can be employed. HotFloorplan uses Hotspot-3.0's [35] grid model with a 50×50 grid resolution. Hotspot's default heat sink model and a starting ambient temperature of 45 ℃ is assumed for all temperature experiments throughout the paper. For 3D floorplans, each die is modeled as two layers – the active silicon and the bulk silicon. The dies are bonded face-to-face (F2F) and the heat sink is placed below the bottom die. A layer of thermal interface material (TIM) is modeled between the bulk silicon of the bottom die and the heat spreader [36]. The thermal parameters for the various layers of the 3D chip are listed in Table 5. The power consumed by data wires between pipeline stages at 90 nm is also considered [37]. Hotspot does not consider interconnect power for thermal modeling. Hence, consistent with other recent evaluations [38], interconnect power is attributed to the units that they connect in proportion to their respective areas. Similar to the methodology in [15], the reduction in area footprint from 3D is assumed to cause a proportional reduction in clock distribution power.

3.2. IPC Impact of Wire Delays

For each of the critical sets of pipeline stages listed in Table 1, we introduce additional wire delays of 2, 4, 6, and 8 cycles. The resulting IPC degradation curves (averaged across the benchmark suite), relative to the baseline processor (that imposes zero inter-block wire delay penalties), are charted in Figure 2. For the single-threaded workloads, it is evident that wire delays between the ALU and data cache have the greatest impact on performance, causing an average slowdown of 20% for a 4-cycle delay. Integer programs are impacted more than FP programs, with *gap*, *gzip*, and *bzip2* exhibiting slowdowns of greater than 40%. As shown in Table 1, delays between the ALU and data cache affect multiple critical loops. The load-to-use loop contributes nearly three-fourth of the 20% observed slowdown, with the remaining attributed to the load-hit speculation loop and L1 miss penalty loop. The load-hit speculation loop also contributes to the second-most critical wire delay, that between the issue queue and ALUs. Since the wakeup and bypass loops are decoupled, a 4-cycle wire delay between the issue queue and ALU only causes a performance degradation of 8%, much lower than the pessimistic 65% degradation reported in [20]. Similarly, because of the decoupled front-end, a 4-cycle wire delay between the branch predictor and I-cache only causes a 2.3% performance loss (instead of the 50% performance loss reported in [20]). To establish confidence in our simulation infrastructure, we modeled the coupled IQ-ALU and front-end in an attempt to reproduce the results in [48]: we observed slowdowns of 68% and 41%, respectively, quite similar to the numbers reported in [48]. The new branch predictor algorithm (indexing with basic block start address instead of basic block end address) affects accuracy by 0.55%. All other wire delays are non-critical and cause slowdowns of less than 5% (for a 4-cycle delay).

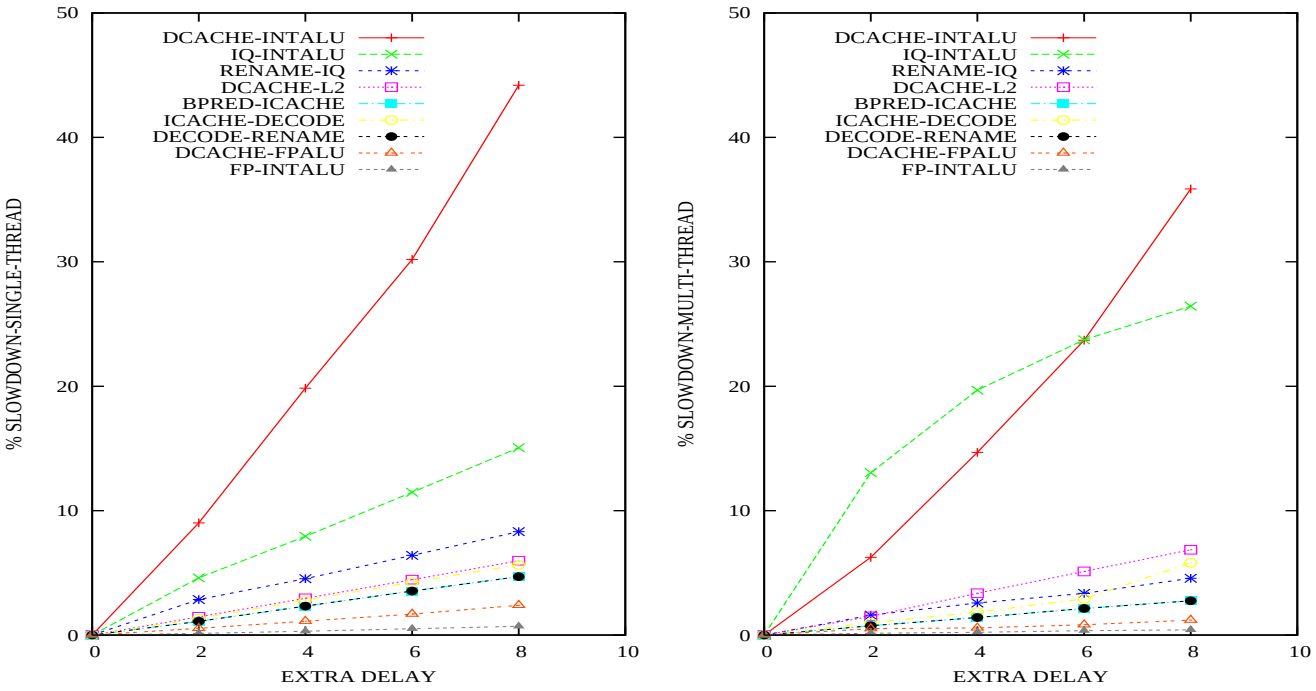


Figure 2: IPC slow-down curves for single-threaded and multi-threaded workloads.

When a floorplanning algorithm evaluates a floorplan with various wire delays between pipeline stages, it must predict the expected overall IPC. If the effects of different wire delays are roughly additive, it is fairly straightforward to predict the IPC of a configuration with arbitrary wire delays between pipeline stages. The predicted theoretical IPC slowdown (relative to the baseline processor with zero inter-block wire delays) for such a processor equals $\sum_i d_i.\mu_i$, where d_i represents each wire delay and μ_i represents the slope of the corresponding slowdown curve in Figure 2. If this hypothesis is true, detailed architectural simulations can be avoided for every floorplan that is considered. To verify this hypothesis, we simulated ten processor configurations with random wire delays (between 0 and 4 cycles) between every pair of pipeline stages. The experimental slowdown closely matches the theoretical slowdown computed according to the slopes of the curves in Figure 2. The maximum and average errors were 4% and 2.1%, respectively. We also repeated our experiments for out-of-order processor models with a range of resources and found little difference in the relative slopes of each slowdown curve.

The graph on the right in Figure 2 shows results for multi-threaded workloads. Since the multi-threaded workloads only include a sub-set of all programs, we normalize the multi-thread slowdowns against the single-thread slowdowns observed for those programs. Hence, it is a reasonable approximation to directly compare the data in the two graphs in Figure 2. For almost all loops, the multi-threaded processor is slightly less sensitive to wire delays because it can find useful work in other threads during stall cycles. The only exception is the IQ-ALU loop. Wire delays in the IQ-ALU loop increase the load-hit speculation penalty. An increase in this delay causes the thread to issue more speculative instructions

– hence wire delays are an impediment to the execution of the co-scheduled thread, not an opportunity to steal idle slots. Further, as this wire delay increases, issue queue occupancy increases and since this is a shared resource, it further inhibits the co-scheduled thread. We have verified that the multi-threaded results do not influence our 2D/3D floorplans greatly, so the rest of the paper only discusses single-thread workloads.

3.3. Floorplanning Results

Critical loop	Weight	Delay for optimal 2D floorplan 4X wires (4 GHz)	Delay for optimal 2D floorplan 8X wires (2 GHz)	Delay for optimal 3D floorplan 4X wires (4 GHz)	Delay for optimal 3D floorplan 8X wires (2Ghz)
DCACHE-INTALU	18	1	1	0	0
DCACHE-FPALU	1	3	1	1	1
BPRED-ICACHE	2	1	1	1	1
IQ-INTALU	6	1	1	1	1
FP-INTALU	1	2	1	1	1
DECODE-RENAME	2	1	1	1	1
RENAME-IQ	4	1	1	1	1
DCACHE-L2	2	1	1	1	1
DECODE - ICACHE	2	2	1	1	1

Table 6: Weights for the different pairs of blocks and the corresponding wire delays (in cycles) for the optimal 2D and 3D floorplans for a highly wire-constrained model (4X wires at 4 GHz frequency) and a marginally wire-constrained model (8X wires at 2 GHz frequency).

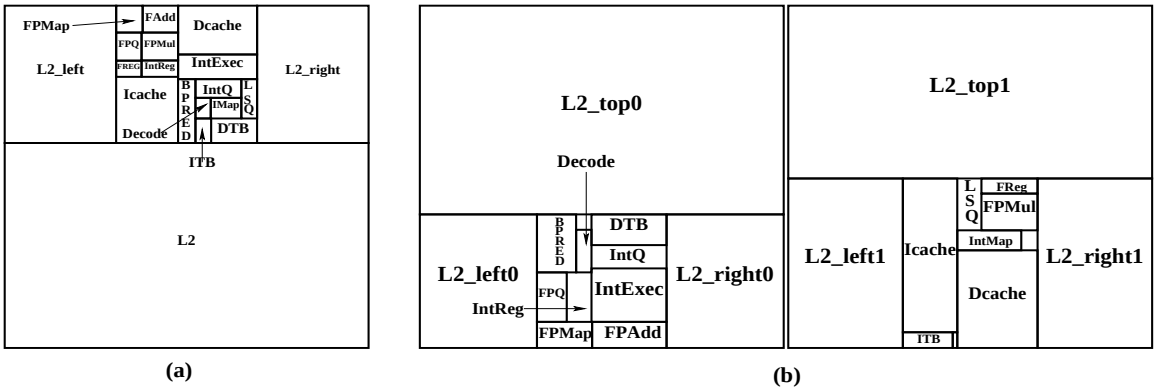


Figure 3: Optimal 2D and 3D floorplans.

The microprocessor model fed to HotFloorplan is very similar to the Alpha 21264 [23] – the same microarchitectural blocks and relative sizes are assumed. The objective function for the floorplanning algorithm includes a term for IPC penalty. The slopes of the slowdown

curves in Figure 2 are used to compute the $IPC\_penalty$ weights for each set of wires. These weights are listed in Table 6. To estimate the performance of each floorplan, we determine the distances between the centers of interacting blocks and compute wire latencies for two different types of wires – fast global wires on the 8X metal plane and semi-global wires on the 4X plane[1]. These latencies are converted to cycles for two different clock speed assumptions – 2 GHz and 4 GHz. The data in Table 6 shows the corresponding cycles required for each communication in the most wire-constrained model (wires are implemented on the slower 4X metal plane and a fast clock speed of 4 GHz is assumed) and the least wire-constrained model (wires are implemented on the faster 8X plane and a clock speed of 2 GHz is assumed) for the optimal 2D floorplan. If the wire delay between blocks is less than 1 FO4, we assume that the delay can be somehow absorbed in the previous pipeline stage and no additional cycles of wire delay are introduced. The L2 latency is determined by adding the wire delay between the L1 cache and the nearest L2 bank to the 30-cycle L2 access time.

We observe that the optimal 2D floorplan (shown in Figure 3(a)) co-locates the units that are involved in the most critical wire delays (DCache-IntALU, IQ-IntALU). Because the critical wire delays are minimized, the IPC slowdown incurred by all the introduced wire delays is only 11.5% in the most wire-constrained model and 10% in the least wire-constrained model. This result indicates that it is fairly easy to minimize wire delays between critical units even in two dimensions. For the optimal 2D floorplan above, wire delays impose a performance penalty of 11.5% at most and this represents an upper bound on the performance improvement that 3D can provide. Figure 3(b) and Table 6 also show the optimal 3D floorplan and its corresponding communication latencies. The wire delays impose a performance penalty of 7% at most. Hence, for a traditional out-of-order superscalar processor, the stacking of microarchitectural structures in three dimensions enables a performance improvement of at most 4.5%.

Model	2D	3D	Difference
Most-constrained (Peak)	81.4	94.1	12.7
Least-constrained (Peak)	69.2	75.3	6.1
Most-constrained (Avg)	75.7	83.5	7.8
Least-constrained (Avg)	66.7	70.5	3.8

Table 7: Optimal floorplan temperatures in ℃

According to our floorplan models, the peak temperatures for the 3D floorplan are on an average 12.7 ℃ and 6.1 ℃ higher than the 2D floorplan for the most and least wire-constrained models, respectively(Table 7). We estimated the power dissipated by inter-block wires in 2D and 3D based on the number of maximum bits being transferred between blocks, the distance traveled, power per unit length for 4X and 8X wires at 90 nm technology, and

1. The processor is assumed to have four types of metal layers, 1X, 2X, 4X, and 8X, with the notation denoting the relative dimensions of minimum-width wires [39]. 1X and 2X planes are used for local wiring within circuit blocks.

an activity factor of 0.5. In addition to the 50% reduction in clock distribution power, we observed a 39% reduction in power for inter-block communication. We acknowledge that it is difficult to capture inter-block control signals in such a quantification, so this is a rough estimate at best.

Our primary conclusion from the first half of the paper is that the potential for performance improvement with the *stacking* approach is marginal for simple superscalar cores. However, the associated design/complexity cost is low and energy per instruction is reduced.

Our conclusions differ from those drawn in the study by Black et al. [6]. That study reports a 15% performance improvement by implementing a Pentium4 core in 3D. This difference can be attributed to two sources: (i) The Pentium4 2D layout places the SIMD unit between the FP register file and FP ALUs. This introduces a 1-cycle delay between the FP register file and FP ALUs, modeled as a 2-cycle increase in the latency of all FP instructions. The move to 3D eliminates this wire delay and improves performance by 4%. If the latency of all FP instructions is reduced by 2 cycles in our simulation infrastructure, we too observe a similar 3% performance improvement (for the SPEC2k FP benchmarks). However, this aspect is left out of the results shown above. Firstly, as explained in Section 2.1, a delay between the register file and ALU should not introduce stall cycles between dependent instructions if full bypassing is provided. Secondly, the FP ALUs and FP register file are in close proximity in our 2D layout. These observations do highlight the point that a 2D industrial implementation may have some inefficiencies (for example, inability to provide full bypassing or inability to co-locate certain structures) that can be elided with a 3D layout. (ii) The Pentium4 has a small store queue and is sensitive to post-retirement pipeline stages involving the store instruction. By eliminating post-retirement wire delay and releasing the store queue entry sooner, a 3% performance improvement is reported in [6]. In our simulation environment, if we implement a 14-entry store queue and release an entry 30 cycles after the store retires, we observe a 3% improvement if the post-retirement delay is reduced by 30%. This phenomenon disappears if the store queue size is eliminated as a bottleneck (by increasing its size to at least 24). We assume that the store queue size will not be a bottleneck and exclude this aspect from our simulation results. Our study is more pessimistic about the potential of 3D because our pipeline model has balanced resources, is simpler and shorter (perhaps more indicative of future cores), and takes a rosy view of the efficiency of a 2D layout.

4. 3D Layouts for Clustered Architectures

The high-level insight drawn from the previous section can be summarized as follows: (i) delays between the integer ALUs and data cache are critical and must be minimized, and (ii) it is fairly straightforward to minimize these delays in traditional 2D superscalar processors. While 3D-stacking may offer little benefit for such processors, it may hold more promise for ILP in an aggressive superscalar with many resources. A clustered architecture is an example of a single-thread[2] complexity-effective processor implementation that leverages a number

2. Clustered architectures can be multi-threaded [40, 41, 42] and are attractive as they are capable of high clock speeds, high ILP, and high thread-level parallelism.

of distributed resources for high ILP [43, 44, 45, 42, 46]. Since clustered architectures incorporate multiple distributed ALUs and data cache banks, they are more limited by wire delays and likely to benefit from 3D. The rest of the paper quantifies the potential of 3D for a specific dynamically scheduled implementation of a clustered architecture, although, we expect the gathered insight to also apply to other partitioned architecture implementations.

4.1. Baseline Clustered Architecture

Many prior papers (such as [43, 44, 45, 47, 42, 46]) have shown that a clustered microarchitecture is a complexity-effective implementation of a high-ILP processor capable of supporting a large window of in-flight instructions. The Alpha 21264 is a commercial example of such a design, where integer execution resources are partitioned across two clusters [23].

The clustered architecture employs small computation units (clusters) that can be easily replicated on the die. An interconnect fabric enables communication between clusters. Each cluster consists of a small issue queue, physical register file, and a limited number of functional units with a single cycle bypass network among them. Dependence chains can execute quickly if they only access values within a cluster. The small sizes of structures within each cluster enable high clock speed and low design complexity.

Our baseline clustered processor model incorporates state-of-the-art features described in recent literature [48, 49, 44, 50, 51] which are summarized below. As shown in Figure 4, instruction fetch, decode, and dispatch (register rename) are centralized in our processor model. During register rename, instructions are assigned to one of eight clusters. The instruction steering heuristic is based on Canal et al.'s ARMBS algorithm [44] and attempts to minimize load imbalance and inter-cluster communication. For every instruction, we assign weights to each cluster to determine the cluster that is most likely to minimize communication and issue-related stalls. Weights are assigned to a cluster if it produces input operands for the instruction. A cluster also receives weights depending on the number of free issue queue entries within the cluster. Each instruction is assigned to the cluster that has the highest weight according to the above calculations. If that cluster has no free register and issue queue resources, the instruction is assigned to a neighboring cluster with available resources. If an instruction requires an operand that resides in a remote register file, the register rename stage inserts a "copy instruction" [44] in the producing cluster so that the value is moved to the consumer's register file as soon as it is produced. These register value communications happen over longer global wires and can take up a few cycles. Aggarwal and Franklin [48] show that a crossbar interconnect performs the best when connecting a small number of clusters (up to four), while a hierarchical interconnect (such as the one shown in Figure 4) performs better for a large number of clusters.

Cache Organization Strategies: In this paper, we consider centralized and distributed versions of the L1 data cache [49, 52, 50, 51]. Load and store instructions are assigned to clusters, where effective address computation happens. The effective addresses are then sent to the corresponding LSQ and L1 data cache bank. For a centralized cache organization, a single LSQ checks for memory dependences before issuing the load and returning the word back to the requesting cluster. When dispatching load instructions, the steering heuristic assigns more weights to clusters that are closest to the centralized data cache.

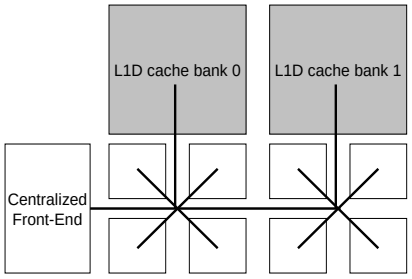


Figure 4: Baseline 2D implementation of the 8-cluster system.

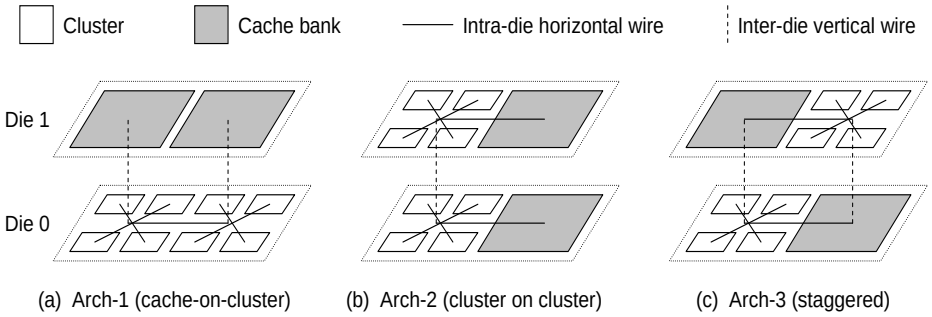


Figure 5: Block diagrams for 3D organizations of the 8 cluster system.

As examples of decentralized cache organizations, we consider replicated and word-interleaved caches. In a replicated cache, each cache bank maintains a copy of the L1 data cache. This ensures that every cluster is relatively close to all of the data in the L1 cache. However, in addition to the high area overhead, every write and cache refill must now be sent to every cache bank. An LSQ at every cache bank checks for memory dependences before issuing loads. A word-interleaved cache distributes every cache line among the various cache banks (for example, all odd words in one bank and even words in another bank). This ensures that every cluster is relatively close to some of the data in the L1 cache. Word-interleaved caches have larger capacities than replicated caches for a fixed area budget. Once the effective address is computed, it is sent to the corresponding LSQ and cache bank. Load instructions must be steered to clusters that are in close proximity to the appropriate cache bank. Since the effective address is not known at dispatch time, a predictor is required. We experimented with various bank predictors and were unable to achieve an accuracy higher than 70%. The instruction steering heuristic performs best if it dis-regards the bank prediction and attempts to optimize the other criteria. A mechanism [51] is required to ensure that memory dependences are not violated. When a store is dispatched, each LSQ is assigned a dummy entry for that store, preventing subsequent loads from issuing. Once the store address is known, it is communicated to the corresponding LSQ and the other LSQ removes the dummy entry. Thus, both decentralized caches suffer from the problem that stores have to be broadcast to all LSQs.

4.2. 3D Layouts for Clustered Architectures

In this sub-section, we describe the three most interesting layouts for the relative placement of cache banks and clusters in 3D (shown in Figure 5). For the rest of this discussion, we assume that (i) two dies are bonded face-to-face (F2F [53]), (ii) each cache bank has the same area as a set of four clusters, and (iii) the system has eight clusters and two cache banks. The floorplanning principles can also be extended to greater numbers of clusters, cache banks, and dies. The differentiating design choices for the three architectures in Figure 5 are: (i) How close is a cluster to each cache bank? (ii) Which communication link exploits the low-latency inter-die via? These choices impact both temperature and performance.

Architecture 1 (cache-on-cluster):

In this architecture, all eight clusters are placed on the lower device layer (die 0) while the data cache banks are placed on the upper device layer (die 1). The heat sink and spreader are placed below the lower device layer (close to the relatively hot clusters). The L1 data cache is decentralized and may either be replicated or word-interleaved. The link from each crossbar to the cache banks is implemented with inter-die vias. Inter-die vias are projected to have extremely low latencies and sufficient bandwidth to support communication for 64-bit register values [3]. In such an architecture, communication between two sets of four clusters can be expensive. Such communication is especially encountered for the word-interleaved cache model and for programs with poor register locality. By placing all (relatively hot) clusters on a single die, the rate of lateral heat spreading is negatively impacted. On the other hand, vertical heat spreading is encouraged by placing (relatively) cool cache banks upon clusters.

Architecture 2 (cluster-on-cluster):

This is effectively a rotated variation of Architecture 1. Clusters are stacked vertically, and similarly, cache banks are also stacked vertically. In terms of performance, communication between sets of four clusters is now on faster inter-die vias, while communication between a cluster and its closest cache bank is expensive. In terms of thermal characteristics, the rate of lateral heat spreading on a die is encouraged, while the rate of vertical heat spreading between dies is discouraged.

Architecture 3 (staggered):

Architecture 3 attempts to surround hot clusters with cool cache banks in the horizontal and vertical directions with a *staggered* layout. This promotes the rate of vertical and lateral heat spreading. Each set of four clusters has a link to a cache bank on the same die and a low-latency inter-die link to a cache bank on the other die. Thus, access to cache banks is extremely fast. In a replicated cache, a load always employs the corresponding vertical interconnect to access the cache bank. On the other hand, register communication between sets of four clusters may now be more expensive as three routers must be navigated. However, there are two equidistant paths available for register communication, leading to

3. Inter-die vias have a length of 10μm and a pitch of 5μm [14].

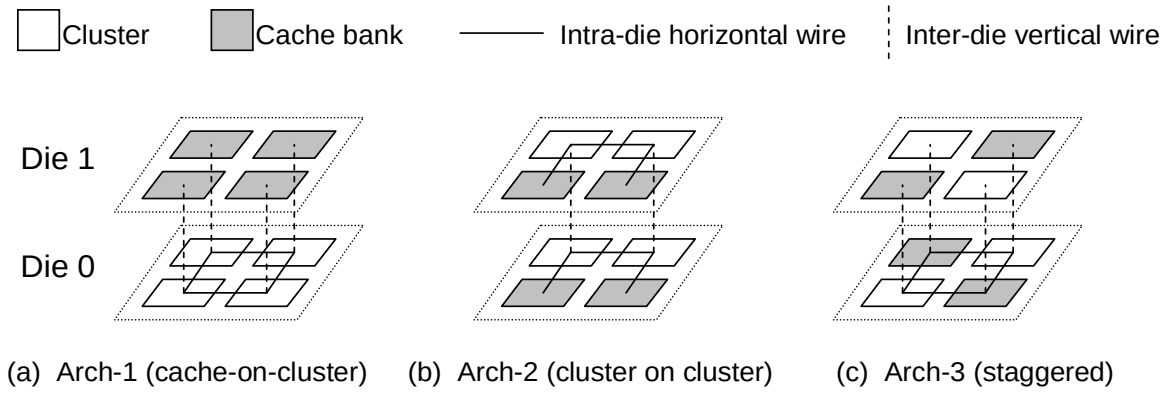


Figure 6: Block diagrams for 3D organizations of the 4 cluster system.

fewer contention cycles. In our experiments, register transfers are alternately sent on the two available paths.

Sensitivity Study:

Most of our evaluation employs a specific 8-cluster 2-bank system to understand how 3D organizations impact performance and temperature characteristics. As future work, we plan to also quantify these effects as a function of number of dies, clusters, cache banks, network characteristics, different resource sizes, etc. For this paper, we repeat our experiments for one other baseline system with four clusters. This helps confirm that our overall conclusions are not unique to a specific processor model. The second processor model has four clusters and each cluster is associated with a cache bank (either word-interleaved or replicated). The clusters are connected with a ring network. Figure 6 illustrates the two-die organizations studied for the 4-cluster system.

5. Results for 3D Layouts

5.1. Methodology

Each cluster in our partitioned architecture has 30 integer and FP registers, 15 integer and FP issue queue entries, and one functional unit of each kind. The latencies of inter-cluster interconnects are estimated based on distances between the centers of microarchitectural blocks in the floorplan. Intra-die interconnects are implemented on the 8X metal plane, and an aggressive clock speed of 4 GHz is assumed. Figures 5 and 6 are representative of the relative sizes of clusters and cache banks (the 8-cluster and 4-cluster systems employ L1 caches with capacities of 64KB and 32KB, respectively). Each crossbar router accounts for a single cycle delay[4]. For the topology in Figure 4, for intra-die interconnects, it takes four cycles to send data between two crossbars, one cycle to send data between a crossbar and cluster, and three cycles to send data between the crossbar and 32KB cache bank.

4. We aggressively assume some form of speculative router pipeline, such as those described in [54, 55].

All vertical inter-die interconnects are assumed to have a single cycle latency due to their extremely short length (10μm [14]). For intra-die interconnects in the 4-cluster organization, the inter-crossbar latency is two cycles. F2F bonding allows a relatively high inter-die via density [14] because of which we assume that the inter-die bandwidth is not a limiting constraint for our experiments. The Wattch power models are employed to compute power consumption of each microarchitectural block. The contribution of leakage to total chip power is roughly 20%. Interconnect power (summarized in Table 8) is based on values for 8X minimum-width wires [56] and a generic Network-on-Chip router [14].

Parameter	Value
Router+Xbar Area	0.375 mm^2 [14]
Router+Xbar Power	119.55mW [14]
Wire Power/Length(mW/mm)	1.25 (8X), 1.40 (4X)

Table 8: Interconnect parameters.

5.2. IPC Analysis

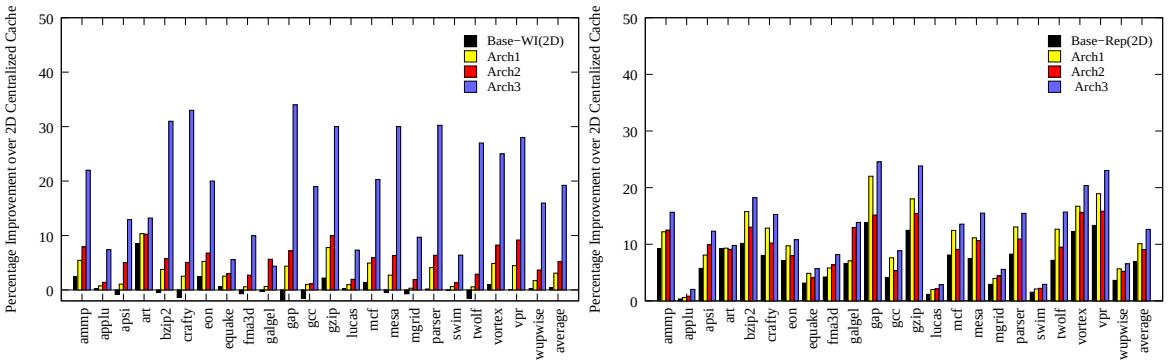
The primary difference between Architectures 1/2/3 (Figure 5) is the set of links that are implemented as inter-die vias. Hence, much of our IPC results can be explained based on the amount of traffic on each set of links. In a word-interleaved cache, nearly half the cache accesses are to the remote cache bank through the inter-crossbar interconnect. In a replicated cache organization, all load requests are sent to the local cache bank. About half as many register transfers are sent on the inter-crossbar interconnect between clusters. Table 9 shows the average network latencies experienced by loads and register transfers in the most relevant 8-cluster architectures.

For all of our results, we fix the 2D 8-cluster system with a centralized cache as the baseline. A 2D system with a word-interleaved cache performs only 1% better than the baseline, mostly because of the large number of remote cache bank accesses. A 2D system with a replicated cache performs about 7% better than the baseline. The replicated cache performs better in spite of having half the L1 data cache size – the average increase in the number of L1 misses in moving from a 64KB to a 32KB cache is 0.88%. A replicated cache allows instructions to not only be close to relevant data, but also close to relevant register operands. However, store addresses and data are broadcast to both cache banks and data is written into both banks (in a word-interleaved organization, only store addresses are broadcast to both banks).

Figure 7 shows IPC improvements for word-interleaved and replicated cache organizations over the 2D baseline. The word-interleaved organizations are more communication-bound and stand to gain much more from 3D. The staggered architecture-3 performs especially well (19.2% better than the baseline) as every cluster is relatively close to both cache banks and multiple network paths lead to fewer contention cycles. Architecture-2 performs better than Architecture-1 because it reduces the latency for register traffic, while slowing down access for local cache banks. The opposite effect is seen for the replicated cache organizations because Architecture-2 slows down access for all loads (since every load accesses the local bank).

Access type	Word-int 3D – arch1	Word-int 3D – arch2	Word-int 3D – arch3	Replicated 3D – arch1	Replicated 3D – arch2	Replicated 3D – arch3
Local load accesses	4.62	6.21	4.94	4.12	5.24	4.13
Remote load accesses	9.89	9.10	7.54	0	0	0
Inter-xbar register traffic	8.67	7.68	6.01	8.55	7.10	5.53

Table 9: Average network latencies (in cycles) for different types of interconnect messages.



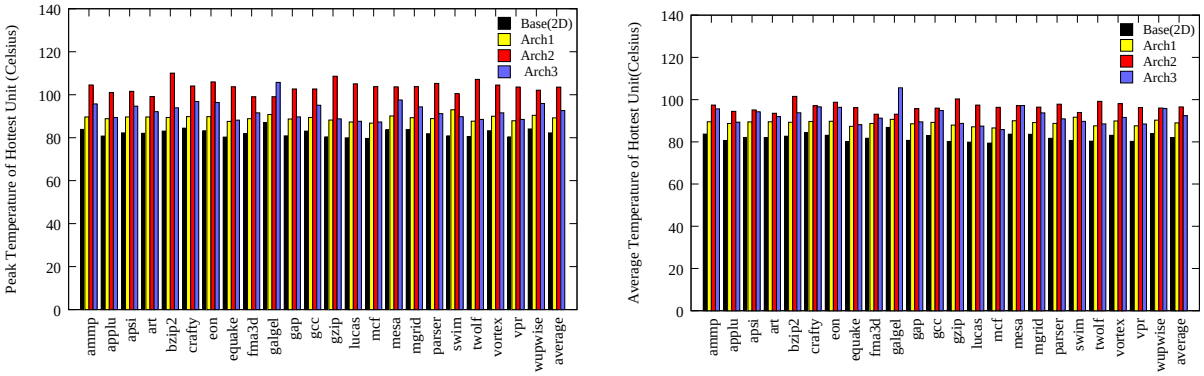
a. IPC improvements for 3D word-interleaved architectures.

b. IPC improvements for 3D replicated architectures.

Figure 7: 3D clustered architectures, relative to a 2D organization with a centralized cache.

With the replicated cache, architecture-3 is similar to architecture-1 as regards cache access, but imposes greater link latency for inter-cluster register communication. Because there are multiple paths for register communication, architecture-3 imposes fewer contention cycles. As can be seen in Table 9, the average total latency encountered by register transfers is lowest for architecture-3, for both word-interleaved and replicated organizations. The net result is that architecture-3 performs best for both cache organizations. The move to 3D causes only a 5% improvement for a replicated cache organization, while it causes an 18.8% improvement for the word-interleaved organization. For the architecture-3 model, the word-interleaved and replicated organizations have similar latencies for instructions (Table 9), but the word-interleaved organization has twice as much L1 cache capacity. It is interesting to note that an organization such as the word-interleaved cache, which is quite un-attractive in 2D has the best performance in 3D (arch-3). It out-performs the best-performing 2D organization (with a replicated cache) by 11%.

The conclusions from our sensitivity analysis with a 4-cluster organization are similar. Compared to a 2D baseline with a centralized cache, the 3D word-interleaved architectures 1, 2, and 3 yield an improvement of 8%, 9%, and 15%, respectively. The 3D replicated architectures 1, 2, and 3 yield improvements of 9%, 13%, and 15%, respectively. The move from 2D to 3D yields an improvement of 9.2% for the word-interleaved and 8% for the replicated cache organizations.



a. Peak temperature of hottest on-chip unit (celsius).

b. Average temperature of hottest on-chip unit(Celsius).

Figure 8: Peak and Average Temperatures of hottest on-chip structures.

The primary benefit of 3D is that cache banks can be placed close to clusters, allowing high performance. The architectures that place cache banks close to clusters also have favorable thermal characteristics. For the 8-cluster system, Figure 8 shows the peak and average temperatures of the hottest on-chip unit (typically one of the issue or load/store queues). A similar profile is also observed for the 4-cluster system and the profile is largely insensitive to the choice of word-interleaved or replicated banks. Architectures 1 and 3 that stack cache upon cluster are roughly 11 ℃ cooler than architecture-2 that stacks cluster upon cluster. Architecture-2 is perhaps most indicative of the temperature characteristics of the *folding* approach, where similar structures are stacked vertically. Thus, staggered architecture 3 not only provides the highest performance, but also limits the increase in temperature when moving to 3D (10 ℃ higher than the 2D chip). The lateral heat spreading effect played a very minor role in bringing down architecture 3's temperature – in fact, it was hotter than architecture 1 because of its higher IPC and power density.

The 10 ℃ temperature rise by moving to 3D is consistent with data reported in other studies that stack cache upon a computational unit [27,30]. The results in [6] show a 5 ℃ temperature increase when stacking an L2 cache upon a CPU. This difference can be attributed to the lower power density of an L2 cache. On setting the power density of our L1 cache equal to the power density of our L2 cache (0.015 W/mm^2), we observed a 5.7 ℃ increase in temperature.

3D stacking enables higher performance for the clustered architecture, but is accompanied by a temperature increase. In order to make a fair comparison, we set a constant thermal constraint for all processor models. If the peak temperature for the processor exceeds a given threshold, dynamic frequency scaling (DFS) is employed until the temperature is lowered. DFS was chosen because it entails negligible overhead and is already a part of most modern commercial processors. Every 10,000 cycles, the frequency is lowered in increments of 10% if the temperature exceeds a threshold of 70 ℃. Peak frequency is re-instated if the temperature drops to 10 ℃ less than the thermal threshold.

The 3D architectures trigger many more thermal emergencies and execute at lower frequencies. With dynamic thermal management included, the loss in performance for the

best-performing 2D organization and the best-performing 3D architecture-3 was 4% and 10%, respectively. As a result, the best-performing 3D organization now out-performs the best-performing 2D organization by only 5%.

The results in this section indicate that the 3D *stacking* approach can improve ILP for aggressive superscalar implementations. We observed a 11% IPC improvement for the described 8-cluster model and this improvement can be boosted to 15% if we aggressively assume zero-cycle delays for inter-die interconnects (instead of the single cycle delay assumed in the above experiments). However, some of this benefit disappears if we limit the processor to a fixed thermal constraint. In order to mine the available ILP, the chip will have to be packaged for a higher cooling capacity.

6. Conclusions

3D technology has the potential to improve microprocessor performance, power, and cost in different ways. This paper focuses on the *stacking* approach, where each pipeline stage is a 2D circuit and 3D is leveraged to reduce wire lengths between pipeline stages. While recent studies have shown that such 3D implementations of complex pipelines can yield 15% performance improvements [15], we show that simpler pipelines stand to gain less from 3D. Few wire delays in traditional superscalars impose a non-trivial performance penalty and 2D layouts can mitigate their effects with good floorplanning. A partitioned architecture is an example of a high-ILP processor that suffers from costly wire delays in a 2D planar implementation. We show that a word-interleaved L1 cache with a staggered 3D placement performs 11% better than the best-performing 2D layout, while also causing a relatively low (10 ℃) increase in temperature. However, if we maintain a constant thermal constraint, some of this performance benefit is lost. The key contributions of the paper can be summarized as follows:

- This work is the first to integrate many varied aspects (loop analysis, automated floorplanning, distributed caches) in determining the benefit of 3D for single core performance.
- We present the most comprehensive analysis of the impact of wire delays on critical processor loops (including various pipeline optimizations and SMT cores).
- This is the first body of work to carry out a detailed design space exploration of 3D clustered architectures. We conclude that the staggered cluster-cache layout provides the best performance and temperature.
- Unlike most prior work, our results are largely negative. We therefore argue that the research community should moderate their enthusiasm for the performance potential of 3D for a single core.

We conclude that unless aggressive high-ILP processors are designed with superior cooling capacities, it may be difficult to leverage the 3D stacking approach for higher single-core performance. It is too early to tell if the 3D folding approach is perhaps a better path for higher ILP. 3D technology continues to hold promise for other applications, such as reducing inter-core latencies in a CMP, "snap-on" analysis engines, and stacked cache/memory hierarchies for CMPs. We view these avenues as exciting future work.

Acknowledgements

This work was supported in parts by NSF grant CCF-0430063 and NSF CAREER Award CCF-0545959. We thank the anonymous reviewers and our shepherd, Dr. Alvin Lebeck, for many helpful suggestions to improve the work. We also thank Abhishek Ranjan for his contribution to early parts of this work.

References

[1] Semiconductor Industry Association, "International Technology Roadmap for Semiconductors 2005." `http://www.itrs.net/Links/2005ITRS/Home2005.htm`.

[2] N. Magen, A. Kolodny, U. Weiser, and N. Shamir, "Interconnect Power Dissipation in a Microprocessor," in *Proceedings of System Level Interconnect Prediction*, February 2004.

[3] Y. Xie, G. Loh, B. Black, and K. Bernstein, "Design Space Exploration for 3D Architectures," *ACM Journal of Emerging Technologies in Computing Systems*, vol. 2(2), pp. 65–103, April 2006.

[4] J. Rattner, "Predicting the Future," 2005. Keynote at Intel Developer Forum (article at `http://www.anandtech.com/tradeshows/showdoc.aspx?i=2367&p=3)`.

[5] Samsung Electronics Corporation, "Samsung Electronics Develops World's First Eight-Die Multi-Chip Package for Multimedia Cell Phones," 2005. (Press release from `http://www.samsung.com`).

[6] S. Mysore, B. Agrawal, N. Srivastava, S. Lin, K. Banerjee, and T. Sherwood, "Introspective 3D Chips," in *Proceedings of ASPLOS-XII*, October 2006.

[7] K. Puttaswamy and G. Loh, "Implementing Caches in a 3D Technology for High Performance Processors," in *Proceedings of ICCD*, October 2005.

[8] K. Puttaswamy and G. Loh, "Dynamic Instruction Schedulers in a 3-Dimensional Integration Technology," in *Proceedings of GLSVLSI*, April 2006.

[9] K. Puttaswamy and G. Loh, "Implementing Register Files for High-Performance Microprocessors in a Die-Stacked (3D) Technology," in *Proceedings of ISVLSI*, March 2006.

[10] K. Puttaswamy and G. Loh, "The Impact of 3-Dimensional Integration on the Design of Arithmetic Units," in *Proceedings of ISCAS*, May 2006.

[11] P. Reed, G. Yeung, and B. Black, "Design Aspects of a Microprocessor Data Cache using 3D Die Interconnect Technology," in *Proceedings of International Conference on Integrated Circuit Design and Technology*, May 2005.

[12] Y.-F. Tsai, Y. Xie, N. Vijaykrishnan, and M. Irwin, "Three-Dimensional Cache Design Using 3DCacti," in *Proceedings of ICCD*, October 2005.

[13] G. Loi, B. Agrawal, N. Srivastava, S. Lin, T. Sherwood, and K. Banerjee, "A Thermally-Aware Performance Analysis of Vertically Integrated (3-D) Processor-Memory Hierarchy," in *Proceedings of DAC-43*, June 2006.

[14] F. Li, C. Nicopoulos, T. Richardson, Y. Xie, N. Vijaykrishnan, and M. Kandemir, "Design and Management of 3D Chip Multiprocessors Using Network-in-Memory," in *Proceedings of ISCA-33*, June 2006.

[15] B. Black, M. Annavaram, E. Brekelbaum, J. DeVale, L. Jiang, G. Loh, D. McCauley, P. Morrow, D. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb, "Die Stacking (3D) Microarchitecture," in *Proceedings of MICRO-39*, December 2006.

[16] M. Awasthi and R. Balasubramonian, "Exploring the Design Space for 3D Clustered Architectures," in *Proceedings of the 3rd IBM Watson Conference on Interaction between Architecture, Circuits, and Compilers*, October 2006.

[17] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel, "The Microarchitecture of the Pentium 4 Processor," *Intel Technology Journal*, vol. Q1, 2001.

[18] E. Borch, E. Tune, B. Manne, and J. Emer, "Loose Loops Sink Chips," in *Proceedings of HPCA*, February 2002.

[19] J. Cong, A. Jagannathan, Y. Ma, G. Reinman, J. Wei, and Y. Zhang, "An Automated Design Flow for 3D Microarchitecture Evaluation," in *Proceedings of ASP-DAC*, January 2006.

[20] K. Sankaranarayanan, S. Velusamy, M. Stan, and K. Skadron, "A Case for Thermal-Aware Floorplanning at the Microarchitectural Level," *Journal of ILP*, vol. 7, October 2005.

[21] E. Sprangle and D. Carmean, "Increasing Processor Performance by Implementing Deeper Pipelines," in *Proceedings of ISCA-29*, May 2002.

[22] G. Reinman, T. Austin, and B. Calder, "A Scalable Front-End Architecture for Fast Instruction Delivery," in *Proceedings of ISCA-26*, May 1999.

[23] R. Kessler, "The Alpha 21264 Microprocessor," *IEEE Micro*, vol. 19, pp. 24–36, March/April 1999.

[24] W. Liao and L. He, "Full-Chip Interconnect Power Estimation and Simulation Considering Concurrent Repeater and Flip-Flop Insertion," in *Proceedings of ICCAD*, 2003.

[25] C. Long, L. Simonson, W. Liao, and L. He, "Floorplanning Optimization with Trajectory Piecewise-Linear Model for Pipelined Interconnects," in *Proceedings of DAC*, 2004.

[26] Y. Liu, A. Shayesteh, G. Memik, and G. Reinman, "Tornado Warning: The Perils of Selective Replay in Multithreaded Processors," in *Proceedings of ICS*, June 2005.

[27] M. Ekpanyapong, J. Minz, T. Watewai, H. Lee, and S. Lim, "Profile-Guided Microarchitectural Floorplanning for Deep Submicron Processor Design," in *Proceedings of DAC-41*, June 2004.

[28] W. Hung, Y. Xie, N. Vijaykrishnan, C. Addo-Quaye, T. Theocharides, and M. Irwin, "Thermal-Aware Floorplanning using Genetic Algorithms," in *Proceedings of ISQED*, March 2005.

[29] D. Burger and T. Austin, "The Simplescalar Toolset, Version 2.0," Tech. Rep. TR-97-1342, University of Wisconsin-Madison, June 1997.

[30] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," in *Proceedings of ASPLOS-X*, October 2002.

[31] D. Tullsen, S. Eggers, J. Emer, H. Levy, J. Lo, and R. Stamm, "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," in *Proceedings of ISCA-23*, May 1996.

[32] W. Hung, G. Link, Y. Xie, N. Vijaykrishnan, and M. Irwin, "Interconnect and Thermal-Aware Floorplanning for 3D Microprocessors," in *Proceedings of ISQED*, March 2006.

[33] D. Wong and C. Liu, "A New Algorithm for Floorplan Design," in *Proceedings of the 23rd ACM/IEEE conference on Design automation*, pp. 101–107, 1986.

[34] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," in *Proceedings of ISCA-27*, pp. 83–94, June 2000.

[35] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, and K. Sankaranarayanan, "Temperature-Aware Microarchitecture," in *Proceedings of ISCA-30*, pp. 2–13, 2003.

[36] K. Puttaswamy and G. Loh, "Thermal Analysis of a 3D Die-Stacked High-Performance Microprocessor," in *Proceedings of GLSVLSI*, April 2006.

[37] K. Banerjee and A. Mehrotra, "A Power-optimal Repeater Insertion Methodology for Global Interconnects in Nanometer Designs," *IEEE Transactions on Electron Devices*, vol. 49, pp. 2001–2007, November 2002.

[38] W.-L. Hung, G. Link, Y. Xie, N. Vijaykrishnan, and M. J. Irwin, "Interconnect and thermal-aware floorplanning for 3d microprocessors," *isqed*, vol. 0, pp. 98–104, 2006.

[39] R. Kumar, V. Zyuban, and D. Tullsen, "Interconnections in Multi-Core Architectures: Understanding Mechanisms, Overheads, and Scaling," in *Proceedings of the 32nd ISCA*, June 2005.

[40] J. Collins and D. Tullsen, "Clustered Multithreaded Architectures – Pursuing Both IPC and Cycle Time," in *Proceedings of the 18th IPDPS*, April 2004.

[41] A. El-Moursy, R. Garg, D. Albonesi, and S. Dwarkadas, "Partitioning Multi-Threaded Processors with a Large Number of Threads," in *Proceedings of ISPASS*, March 2005.

[42] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. Keckler, and C. Moore, "Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture," in *Proceedings of ISCA-30*, June 2003.

[43] R. Balasubramonian, S. Dwarkadas, and D. Albonesi, "Dynamically Managing the Communication-Parallelism Trade-Off in Future Clustered Processors," in *Proceedings of ISCA-30*, pp. 275–286, June 2003.

[44] R. Canal, J. M. Parcerisa, and A. Gonzalez, "Dynamic Code Partitioning for Clustered Architectures," *International Journal of Parallel Programming*, vol. 29, no. 1, pp. 59–79, 2001.

[45] K. Farkas, P. Chow, N. Jouppi, and Z. Vranesic, "The Multicluster Architecture: Reducing Cycle Time through Partitioning," in *Proceedings of MICRO-30*, pp. 149–159, December 1997.

[46] M. Taylor, W. Lee, J. Miller, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. Johnson, J. Kim, J. Psota, A. Raraf, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams," in *Proceedings of ISCA-31*, June 2004.

[47] S. Palacharla, N. Jouppi, and J. Smith, "Complexity-Effective Superscalar Processors," in *Proceedings of ISCA-24*, pp. 206–218, June 1997.

[48] A. Aggarwal and M. Franklin, "An Empirical Study of the Scalability Aspects of Instruction Distribution Algorithms for Clustered Processors," in *Proceedings of ISPASS*, 2001.

[49] R. Balasubramonian, "Cluster Prefetch: Tolerating On-Chip Wire Delays in Clustered Microarchitectures," in *Proceedings of ICS-18*, June 2004.

[50] P. Racunas and Y. Patt, "Partitioned First-Level Cache Design for Clustered Microarchitectures," in *Proceedings of ICS-17*, June 2003.

[51] V. Zyuban and P. Kogge, "Inherently Lower-Power High-Performance Superscalar Architectures," *IEEE Transactions on Computers*, March 2001.

[52] E. Gibert, J. Sanchez, and A. Gonzalez, "Effective Instruction Scheduling Techniques for an Interleaved Cache Clustered VLIW Processor," in *Proceedings of MICRO-35*, pp. 123–133, November 2002.

[53] B. Black, D. Nelson, C. Webb, and N. Samra, "3D Processing Technology and its Impact on IA32 Microprocessors," in *Proceedings of ICCD*, October 2004.

[54] R. Mullins, A. West, and S. Moore, "Low-Latency Virtual-Channel Routers for On-Chip Networks," in *Proceedings of ISCA-31*, May 2004.

[55] L.-S. Peh and W. Dally, "A Delay Model and Speculative Architecture for Pipelined Routers.," in *Proceedings of HPCA-7*, 2001.

[56] L. Cheng, N. Muralimanohar, K. Ramani, R. Balasubramonian, and J. Carter, "Interconnect-Aware Coherence Protocols for Chip Multiprocessors," in *Proceedings of ISCA-33*, June 2006.