Enabling Dynamic Heterogeneity Through Core-on-Core Stacking

Vasileios Kontorinis[1,2], Mohammad K.Tavana[3], Mohammad H.Hajkazemi[3], Dean M.Tullsen[2] and Houman Homayoun[3]

[1]Google

[2]Department of Computer Science and Engineering, University of California, San Diego

[3]Department of Electrical and Computer Engineering, George Mason University

ABSTRACT

Future computing platforms will need to be flexible, scalable, and power-conservative, while saving size, weight, energy, etc. Heterogeneous architecture can address these challenges by allowing each application to run on a core that matches resource needs more closely than a one-size-fits-all core. Dynamic heterogeneous architectures can extend these benefits further, allowing the system to construct the right core at run-time for each application, borrowing or freeing resources only as needed by the particular application that is running. The key insight in the described design is that 3D stacking of cores eliminates the fundamental barrier to dynamic heterogeneity, allowing various resources belonging to different cores to be shared at run-time with minimal overhead.

Categories and Subject Descriptors

C.1 [**Processor Architectures**]: General; C.1.3 [**Processor Architectures**]: Other Architecture Styles—*Adaptable architectures, Heterogeneous (hybrid) systems*

General Terms

Design, Performance

Keywords

3D stacking technology, core-on-core stacking, resource pooling, energy efficiency

1. INTRODUCTION

The quest for performance has meant that modern general-purpose cores are out-of-order, superscalar, with large speculative structures and large caches. Although these technologies are capable of boosting the performance of problematic applications, there is a significant cost in area and power. Unfortunately, this cost is typically paid even for less problematic applications. The problem is one of over-provisioning – costly resources such as register files, reorder buffers, instruction queues, load and store queues, and cache memory are each sized to satisfy the most demanding applications. However most applications will run well with smaller amounts of each of these resources, and few, if any, applications need all resources maximally sized.

While very static embedded cores can be highly tuned to an individual application, the more dynamic the workload (e.g., smart phones, mobile processors, server processors) the more the processor core is over-provisioned to provide high general-purpose performance on the variety of applications that might be run on the core. In the multi-core era, homogeneous multicore designs exacerbate the problem, as the over-provisioning is multiplied by the number of cores. Heterogeneous (or asymmetric) multicore designs, conversely, provide multiple cores with different architectural features; because a thread need only find a single core that runs it effectively, no single core need be over-provisioned for all possible applications. The heterogeneous cores greatly increase the likelihood that a single thread finds a core that matches its execution needs. We focus in particular on single-ISA heterogeneous multicores [7, 9] which can dynamically discover the best thread-to-core mapping, and change that mapping across phases of execution, via core migration. Examples of such architectures now include the ARM big.LITTLE processor and the NVIDIA Tegra 3.

Heterogeneous designs, because they eliminate or reduce over-provisioning without necessarily sacrificing performance, enable significant gains in performance per area and performance per Watt [2]. In the Dark Silicon era, homogeneous cores that cannot be turned on are of no use, but heterogeneous or specialized cores still add value. It has been shown, for example, that when resources are constrained, the optimal general-purpose multicore processor is not actually composed of general-purpose cores, but rather is a collection of specialized cores [10].

Processors in which the cores are diverse but the characteristics of each core is fixed at design time we will call *statically heterogeneous.* Another alternative is to configure the core to meet the exact needs of the current application at runtime; we will call this approach a *dynamically heterogeneous multicore processor.* There will always be a cost to this flexibility – a static core that exactly meets the demands of an application will be more efficient than one that must be configured; however, the likelihood of finding a core that exactly meets the demands of a particular thread are far greater in the dynamic case.

Several designs have been proposed that provide some level of dynamic heterogeneity. These proposals include Core Fusion [4], TFlex [6], and WiDGET [18]. In a 2-dimensional plane, however, these designs all struggle with the same tradeoff. Either the core pipelines are each layed out tightly and resources (e.g., registers) that we would like to share across cores are distant, or shareable resources are configured close together significantly compromising the efficiency of each pipeline.

In the research primarily described in this paper, we make the case that the recent advent of 3-D architectures opens an opportunity to overcome the drawbacks of existing dynamically heterogeneous designs [3]. A 3-D architecture is one where there are multiple layers of logic implemented in vertically stacked layers of silicon. Each layer can be connected to circuit elements on other layers right above or below with very low latency, as low as a few picoseconds [19]. For example, if we have cores on multiple layers we can have each pipeline efficiently designed in the 2D plane, yet register file partitions of one core can still be very close (both in distance and access time) to those of another core on a different layer, making sharing or reassignment of those resources feasible.

In this paper we show the benefits of resource sharing and core-on-core stacking architecture for enhancing performance and power efficiency. This work describes a first-cut 3-D architecture where multiple cores are placed vertically one above the other on different layers of logic, with low-latency sharing of certain pipeline resources. The architecture has several attractive features, including vertical sharing of resources, reuse of traditional 2D pipeline designs that still exploit 3D geometries, reduced energy consumption, and improved performance, especially when otherwise unused resources can be reclaimed.

2. BACKGROUND – 3D DIE STACKING

3D die stacking is a recent technological development which makes it possible to create chip multiprocessors using multiple layers of active silicon bonded with low-latency, high-bandwidth, and very dense vertical interconnects [12]. 3D die stacking technology provides very fast communication, as low as a few picoseconds [19], between processing elements residing on different layers of the chip. Three dimensional integration has the potential to address demands for shrinking footprints, reduced power consumption, and enhanced overall performance of the system. Advances in the manufacturing process [15] make fabrication of 3D stacking architectures feasible as an emerging technology and commercially employed solution [1]. Die stacking adds an extra dimension for placing and routing circuit blocks and provides more flexibility for new architectures to improve performance [13, 19].

3D stacking offers flexibility and features which benefit new architectures: As feature sizes continue to shrink, interconnect delays have become a critical bottleneck in chip performance. Wiring buffers often have to be inserted to keep the delays of long wires tractable, but these additional buffers increase power consumption. By providing a third dimension of interconnect, communication latency and power can be substantially reduced. For example, assuming a quad-core AMD Bulldozer in 45nm technology, the registers of one core are essentially 5000 ps in wire delay from another core that might like to share them. Conversely, the distance between register files for vertically stacked cores is closer to 5 ps. Additionally, three dimensional integration offers the potential for significant increases in bandwidth. Communication between blocks across dies can be significantly faster and wider than with the same block across the same chip.

3D technology, and its implications on processor architecture, is still in the early stages of development. A number of design approaches are possible and many have been proposed, from alternating cores and memory/cache [13], to folding a single pipeline across layers [14].

3. HETEROGENEOUS ARCHITECTURE

This section gives a brief overview of proposed heterogeneous architectures.

3.1 Static Heterogeneous Architecture

In a heterogeneous architecture, the various cores of a chip multiprocessor are not all configured the same, enabling core designs that are each optimized for a subset of the applications that might run on the processor. A heterogeneous architecture allows each thread to run on a core that matches its resource needs more closely than a single one-size-fits-all core. Heterogeneous architectures have existed in many forms, including IBM's Cell processor, TI's OMAP, and AMD's Fusion APU; however, those architectures do not share a single ISA, meaning that code must be mapped a priori to particular core types. Kumar, et al. proposed *Single-ISA Heterogeneous Multicore Architectures* [7, 9] to more effectively use the available hardware. That work proposed statically heterogeneous designs – that is, core designs that vary across the processor at design time. Industry has now adopted these heterogeneous designs. In particular, the ARM big.LITTLE combines a Cortex-A15 (Big) with Cortex-A7 (Little) and the Nvidia Tegra 3 has 4 performance cores and a fifth low-power core that can take over when performance is not critical.

3.2 Dynamic Heterogeneous Architecture

Unlike static heterogeneous architecture, dynamic heterogeneous architectures provide more opportunity to map an application to a core which matches its resource needs more closely. Some of the first efforts to provide this kind of heterogeneity include Core Fusion [4], TFlex [6], and WiDGET [18]. Because of the distance problem, each of these architectures restricts sharing of resources to a relatively coarse granularity to try to minimize steering and inter-pipeline communication delays. Core Fusion has the ability to merge all of the resources of small cores into composite cores that are double or quadruple the size. TFlex also combines narrow-width pipeline cores to create wider cores. WiDGET has the ability to borrow execution units from a neighboring pipeline to construct cores with wider execution bandwidth.

With a 3D architecture, we can dynamically pool resources that are potential performance bottlenecks, for possible sharing with neighboring cores. Because we aggregate resources in the vertical dimension, we can share resources at a much finer granularity. Vertically stacked cores can share resources such as register file, functional units, reorder buffer, instruction queue, load and store queue, private L1 cache, and branch predictor. With 3D integration we can design the pipeline traditionally in the 2D plane, yet have poolable resources connected along the 3rd direction on other layers. Thus, in contrast to prior approaches, we can borrow some renaming registers from one core that is not using them to address a specific bottleneck in a second core, while the second core might donate instruction queue entries it is not using to the first.

The dynamically heterogeneous 3D processors we propose provide several key benefits:

(1) They enable software to run on hardware optimized for the execution characteristics of the running code, even for software the original processor designers did not envision.

(2) They enable us to design the processor with compact, lightweight cores without sacrificing general-purpose performance. Modern cores are typically highly over-provisioned [8] to guarantee good general-purpose performance – if we have the ability to borrow the specific resources that a thread needs, the basic core need not be over-provisioned in any dimension.

(3) The processor provides true general-purpose performance, not just adapting to the needs of a variety of applications, but also to both high thread-level parallelism (enabling many area-efficient cores) and low thread-level parallelism (enabling one or a few heavyweight cores).

3.3 Challenges with Heterogeneous Designs

There remain several research challenges with heterogeneous architecture, beyond those addressed in this work. (1) Architecture Diversity: The design space for static heterogeneous architectures is huge, and identifying an optimal design is difficult. Dynamically heterogeneous designs only push this problem down to the runtime, where finding the best configuration dynamically for a particular thread is still a very difficult problem. Even the static design choices (which resources to share, and how to share without adding complexity to the baseline pipeline) for the dynamic heterogeneous architecture are not obvious. (2) Application Mapping: Mapping applications to a heterogeneous architecture to benefit from the diverse core flavor is also a complex problem, particular because different phases of the same application will often prefer different cores or configurations. (3) Programming Model: Programming a heterogeneous architecture can pose several challenges. While this work assumes heterogeneity that is transparent to the programming model, heterogeneity that introduces more specific accelerators and custom architectures will require new programming paradigms which must balance performance, programmability, and portability.

4. DYNAMIC HETEROGENEITY WITH 3D CORE-ON-CORE STACKING

In this section we present our solution for dynamic heterogeneity and demonstrate how core-on-core stacking enables resource sharing and run-time adaptation in this new architecture.

4.1 Baseline Architecture

In this section, we discuss the baseline chip multiprocessor architecture and derive a reasonable floorplan for the 3D CMP. This floorplan is the basis for our initial power, temperature, and performance modeling of various on-chip structures and the processor as a whole.

In this research, we are simply attempting to add a new alternative to the 3D design space. A principal advantage of the dynamically heterogeneous 3D architecture is that it does not change the fundamental pipeline design of 2D architectures, yet still exploits the 3D technology to provide greater energy proportionality and core customization. For comparison purposes, we will compare against a commonly proposed approach which preserves the 2D pipeline design,

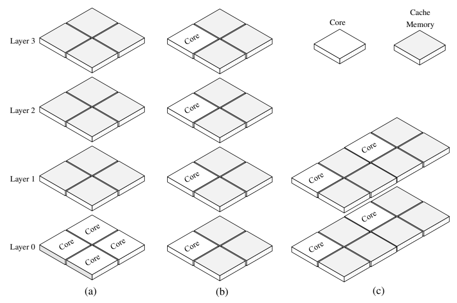

Figure 1: CMP configurations: (a) baseline and (b) 4 layers resource pooling (c) 2 layers resource pooling.

but where core layers enable more extensive cache and memory. For the choice of core we initially study two types of architecture, a high-end architecture which is an aggressive superscalar processor with issue width of 4, and a medium-end architecture which is an out-of-order processor with issue width of 2. The detailed architecture specification for each case has been presented in [3].

4.1.1 Floorplans

For our high-end processor we assume the same floorplan and same area as the Alpha 21264 [5] but scaled down to 45nm technology. For the medium-end architecture we scale down the Alpha 21264 floorplan (in 45nm) based on smaller pipeline components. Moving from 2D to 3D increases power density due to the proximity of the active layers. As a result, temperature is always a concern for 3D designs. Early work in 3D architectures assumed that the best designs sought to alternate hot active logic layers with cooler cache/memory layers. More recent work contradicts that assumption – it is sometimes more important to put the active logic layers as close as possible to the heat sink. Therefore, an architecture that clusters active processor core layers tightly is consistent with this approach. Other research has also exploited this principle.

For the rest of this work we focus on three types of floorplan; the baseline which puts all cores in one layer as shown in Figure 1(b) (thermal-aware) and our proposed floorplans which stack cores vertically across four layers as shown in Figure 1(b) or across two layers as shown in Figure 1(c) (performance-aware). All of these floorplans preserve the traditional 2D pipeline, but each provides a different performance, flexibility, and temperature tradeoff.

The thermal-aware architecture keeps the pipeline logic closest to the heat-sink and does not stack pipeline logic on top of pipeline logic. Conversely, the 3D dynamically heterogeneous configuration stacks pipeline logic on top of pipeline logic, as in other performance-aware designs, gaining increased processor flexibility through resource pooling.

4.2 Motivation for Resource Sharing

Combining 3D die stacking with chip multiprocessors that span multiple layers of active silicon, among other benefits, places more resources within reasonable access time than would otherwise be configured on a single 2D pipeline, thanks to the existence of fast vertical interconnects.

The focus of this work is on resource adaptation in four major delay and performance-critical units – the reorder

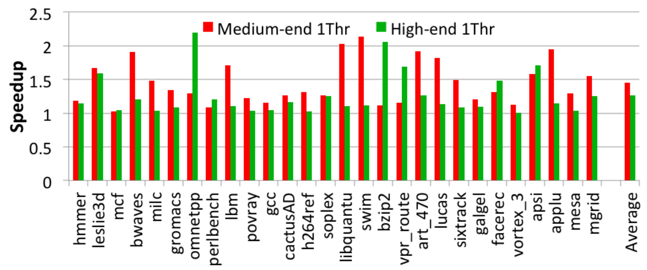

Figure 2: Speedup from increasing resource size in the 3D stacked CMP with medium-end and high-end cores

buffer, register file, load/store queue, and instruction queue. By pooling (allocating resources from a single combined pool to the set of cores that are stacked vertically) just these resources, we create an architecture where an application's scheduling window can grow to meet its runtime demands, potentially benefiting from other applications that do not need large windows.

We assume 4 cores are stacked on top of each other. The maximum gains will be achieved when one, two, or three cores in our 4-core CMP are idle, freeing all of their poolable resources for possible use by running cores. The one-thread case represents a limit study for how much can be gained by pooling, but also represents a very important scenario – the ability to automatically configure a more powerful core when thread level parallelism is low. This does not represent an unrealistic case for this architecture – in a 2D architecture, the cost of quadrupling, say, the register file is high, lengthening wires significantly and moving other key function blocks further away from each other. In this architecture, we are exploiting resources that are already there, the additional wire lengths are much smaller than in the 2D case, and we do not perturb the 2D pipeline layout.

We examine two baseline architectures— a 4-issue high-end core and a 2-issue medium-end core. In Figure 2 we report the speedup for each of these core types when selected resources are quadrupled (when 3 cores are idle). This represents an upper bound for the performance gain with resource sharing across four vertically stacked cores.

In several benchmarks a performance gain of 2X is observed. On average, 45% performance improvement can be achieved for the medium-end processor, and 26% for the high end, by increasing selected window resources. Most importantly, the effect of increased window size varies dramatically by application. This motivates resource pooling, where we can hope to achieve high overall speedup by allocating window resources where they are most beneficial.

5. ARCHITECTURAL SUPPORT, CIRCUIT IMPLEMENTATION, AND TECHNOLOGY MODELING

While resources between stacked cores on different layers are physically close, we still need architectural solutions to allow those resources to be shared (re-allocated from one core to another). We need mechanisms and policies to control the sharing of resources. Architectural solutions are required to allow a partition of a resource to be accessed by other cores. For example, at the architecture level we need to modify register renaming to be able to share a register file between different cores. We also need architectural support to direct the allocation and re-allocation of those partitions. We provide the detailed architectural modifications required to enable resource sharing in various processor units in [3].

The implementation of a heterogeneous IC with multiple cores sharing computing resources requires novel circuits techniques to properly control the signaling between device planes, the synchronization of the data paths, and the power delivery of each plane. Circuit modifications to properly couple two adjacent device planes for resource sharing are described in [3]. Through silicon vias (TSVs), the technology that permits vertical stacking of device planes, can potentially influence the impedance of interconnects between device planes as well as negatively affect the active silicon area available to circuits. Therefore, an electrical analysis of the TSV and the area penalty of using TSVs need to be considered [3]. In addition, circuit techniques and methodologies to distribute power across device planes and synchronize circuit blocks found in disparate technologies are required. Multi-plane power delivery requires that the current demands of each device plane are met while also ensuring no violations of the noise requirements of each tier. The design of the clock distribution network must meet the skew and slew requirements of the data paths across device planes.

In [3] we present the circuit modifications required to enable sharing of the register file, reorder buffer, instruction queue, and load/store queue across different cores. The overall delay added to the ROB or RF due to the required additional multiplexing and decoding is shown to be small, ranging from 0.04 ns to 0.12 ns in 45nm technology, which only effects the clock rate if those resources were already on or near the critical path. The area overhead of the design (e.g., keep out area of TSVs) is found to be lower than 1%.

6. ADAPTIVE ALLOCATION

In addition to the circuit modifications that are necessary to allow resource aggregation across dies, we also need mechanisms and policies to control the pooling or sharing of resources. In devising policies to manage the new shared resources in this architecture, we would like to maximize flexibility; however, design considerations limit the granularity (both in time and space) at which we can partition core resources. Time is actually the easier issue. Because the aggregated structures are quite compact (in total 3D distance), we can reallocate partitions between cores very quickly, within a cycle or cycles. However, to reduce circuit complexity, we expect to physically repartition on a more coarse-grain boundary (e.g., four or eight entries rather than single entries). We experiment with a variety of size granularities for reallocation of pooled resources. Large partitions both restrict the flexibility of pooling and also tend to lengthen the latency to free resources. We also vary how aggressively the system is allowed to reallocate resources; specifically, we explore various static settings for the minimum (MIN) and the maximum (MAX) number of partitions a single core can own, which determine the floor and the ceiling for core resource allocation.

Our baseline allocation strategy exploits two principles. First, we need to be able to allocate resources quickly. Thus, we cannot reassign active partitions, which could take hundreds of cycles or more to clear active state. Instead we actively harvest empty partitions into a free list, from which they can later be assigned quickly. Second, we do not wait

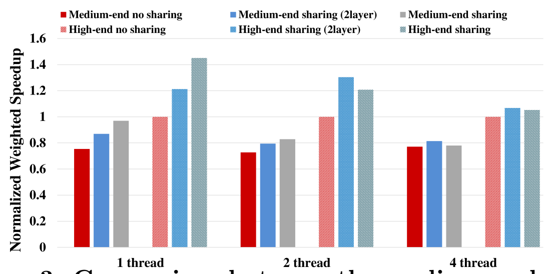

Figure 3: Comparison between the medium-end and the high-end core with and without 3D sharing.

to harvest empty partitions on demand — we grab them immediately when they become free. This works because even if the same core needs the resource again right away, it can typically get it back in a few cycles.

We assume a central arbitration point for the (free) pooled resources. A thread will request additional partitions when a resource is full. If available (on the list of free partitions), and the thread is not yet at its MAX value, those resources can be allocated upon request. As soon as a partition has been found to be empty it is returned to the free list (unless the size of the resource is at MIN). The architecture could adjust MIN and MAX at intervals depending on the behavior of a thread, but this will be the focus of future work – for now we find static values of MIN and MAX to perform well. If two cores request resources in the same cycle, we use a simple round-robin priority scheme to arbitrate.

7. RESULTS

In order to evaluate different resource adaptation policies, we add support for dynamic adaptation to the SMTSIM simulator [17], configured for multicore simulation.The architecture configurations for both medium-end and high-end cores have been presented in [3], in detailed. For power estimation we used McPAT [11] and integrated it with SMTSIM. For temperature calculation we use Hotspot 5.0 [16] and our detailed configuration for thermal analysis has been presented in [3].

7.1 Performance

This section demonstrates the performance advantage of resource pooling. We assume all configurations reported pool resources among vertically stacked cores, whether the workload is four threads or two. Therefore, for the 2-layer floorplan, a total of two cores pool resources together. For the case of a 4-layer floorplan all four cores pool resources together. All cores allocate resources greedily, within the constraints of the MIN and MAX settings. Assuming MIN and MAX are constant over time and the same for all cores we performed a large space exploration (not shown) and found that setting MIN=0.125, MAX=1.0 gives us the best performance results. For the four layer floorplan, our results [3] indicate that while we can get significant performance gains (8-9%) with full utilization (four threads), gains are dramatic when some cores are idle. With two threads we get 26-28% performance for the best policy. Not surprisingly, setting a MIN value to zero, in which case a core can actually give up all resources (for example if it is stalled for an Icache miss) is a bad idea.

To directly compare medium-end and high-end architecture, we show the results for the two architectures (no sharing and sharing with MIN=0.125 and MAX=1.0, for the two layer floorplan and the four layer floorplan) all normalized to the high-end no sharing result, in Figure 3. From this graph we can see that resource pooling makes the medium core significantly more competitive with the high-end. Without sharing, the medium core operates at 75% of the performance of the high end. For the 4-layer floorplan, with pooling and four active threads it operates at 79%, with two active threads, it operates at 83%, and with one active thread, it operates at 97% of the performance of the high-end core. For the 2-layer floorplan, with pooling and two active threads it operates at 81% of the high end core; with one active thread, it operates at 87% of the high-end core. In both floorplans, the larger number of idle cores provide potential to bridge the performance gap between a high performance core and medium core, using resource sharing. Also, comparing the 2-layer and 4-layer floorplan we can observe that for one active thread the 4-layer design provide substantially higher performance compared to a 2-layer design, as expected. However as we increase the number of active threads the performance gap between the 2-layer design and the 4-layer design diminishes. In some cases the 2-layer case outperforms the 4-layer – some static partitioning actually helps. This implies that our simple greedy allocation of partitions is not providing optimal sharing with the larger number of cores, an opportunity for future research. The one case where the difference is significant is with two threads – in this case each thread (in the two layer case) gets all the resources of two cores; however, with four-layer sharing and two threads, one greedy thread can still interfere with the other thread.

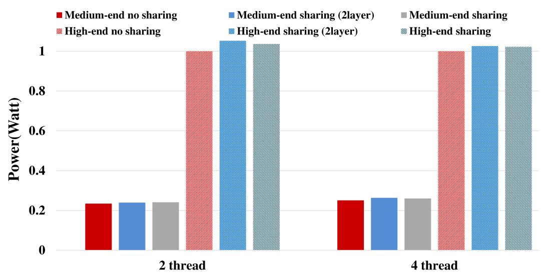

Figure 4: Power consumption per core for MIN=0.125, MAX=1.0 as well as the baseline (no sharing) for 2 and 4 thread workloads. All results normalized to high end core power when there is no resource sharing.

7.2 Power, Temperature, and Energy

Figure 4 shows the power consumption of the studied architectures. The pooling processor architectures pay a small price in power, in large part because of the enhanced throughput. The small additional power overhead is in contrast in some cases to the large performance benefit (in terms of weighted speed up). This is not an obvious result. It happens because weighted speedup weights application speedups equally (rather than over-weighting high-IPC threads, which throughput measures do). Because we get some large speedups on low-IPC threads, we see high average speedup, but smaller increase in total instruction throughput and thus smaller increase in power.

Because of the layout advantages (remember, the baseline processor places all cores right next to the heat sink), the cost in maximum temperature is more significant (Figure 5). Interestingly, the temperature of the medium resource-pooling

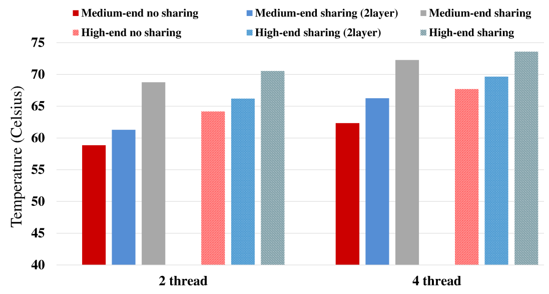

Figure 5: MAX temperature for MIN=0.125, MAX=1 and baseline for 2-thread workloads and 4-thread workloads.

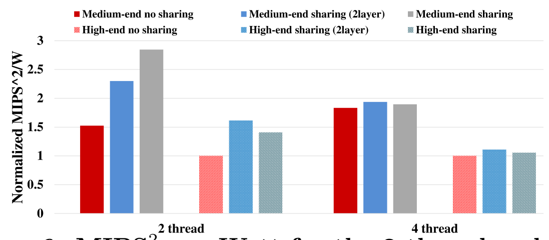

Figure 6: MIPS^2 per Watt for the 2-thread and the 4-thread workloads normalized to the high-end configuration without sharing.

core is comparable to the high-end core. This is because we assume the medium core is laid out tightly, resulting in a slightly higher max temperature for four-thread workloads. For two-thread workloads, the medium resource-pooling core has slightly lower temperature than the high-end core (average 2 degree lower). If the medium core covered the same area as the high-end core, for example, the max temperature would be significantly lower. As Figure 5 shows, in the medium-end architecture the temperature rise is more significant than the high-end architecture, i.e., 10^{o}C compared to 6^{o}C in both 2 thread and 4 thread. This is because we assume a weaker packaging for the medium-end design compared to the high-end design. As expected, by stacking more layers, the temperature rises, however, the 2-layer floorplan adds only 4^{o}C in temperature, while the 4-layer resource pooling increases the temperature by 10^{o}C. Even still, at equal temperature, the more modest cores have a significant advantage in energy efficiency measured in MIPS^2/W (note that MIPS^2/W is the inverse of energy-delay product), as seen in Figure 6. This is a critical result. By outperforming the non-pooling medium core, and approaching the performance in some cases of the large core (due to its just-in-time provisioning of resources), the dynamically heterogeneous medium-end core provides the highest energy efficiency. For high-end cores, the 2-layer resource pooling always provide better energy efficiency compare to 4-layer. For both medium-end and high-end cores the energy efficiency significantly improves with resource pooling when 2 threads are running (up to 2.8x and 1.6x respectively). However, when the load is high (4 thread), the improvement is minor for high-end cores. In other words, more idle cores provide more opportunity to exploit resource sharing and increase energy efficiency.

8. CONCLUSIONS

A dynamically heterogeneous architecture aims to change both the configuration and the quantity of cores available, in response to the running workload. The challenge with existing dynamically heterogeneous design is that they can only configure the resources at coarse granularities, due to the communication distances across pipelines. This paper describes a dynamically heterogeneous 3D stacked architecture which enables very fine-grain reallocation of resources between cores, through resource pooling on a stacked chip multiprocessor architecture. It leverages our current expertise in creating tight 2D pipelines on one layer, while accessing pooled resources of the same type on other layers. By eliminating the need to over-provision each core, modest cores become more competitive with high-performance cores, enabling an architecture that gives up little in performance, yet provides strong gains in energy-delay product over a conventional high-performance architecture.

9. REFERENCES

[1] Tezzaron semiconductor. In *www.tezzaron.com*.
[2] M. Hill and M. Marty. Amdahl's law in the multicore era. *Computer*, 2008.
[3] H. Homayoun et al. Dynamically heterogeneous cores through 3d resource pooling. In *HPCA*, Feb 2012.
[4] E. Ipek et al. Core fusion: Accommodating software diversity in chip multiprocessors. In *ISCA*, June 2007.
[5] R. Kessler et al. The alpha 21264 microprocessor architecture. In *ICCD*, 1998.
[6] C. Kim et al. Composable lightweight processors. In *MICRO*, 2007.
[7] R. Kumar and et. al. Single-ISA Heterogeneous Multi-core Architectures: The Potential for Processor Power Reduction. In *MICRO*, 2003.
[8] R. Kumar and et. al. Conjoined-core chip multiprocessing. In *MICRO*, Dec. 2004.
[9] R. Kumar et al. Single-ISA Heterogeneous Multi-core Architectures for Multithreaded Workload Performance. In *ISCA*, June 2004.
[10] R. Kumar and et. al. Core architecture optimization for heterogeneous chip multiprocessors. In *PACT*, 2006.
[11] S. Li and et. al. Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*, pages 469–480. IEEE, 2009.
[12] G. H. Loh, Y. Xie, and B. Black. Processor design in 3d die-stacking technologies. *IEEE Micro*, 2007.
[13] N. Madan et al. Optimizing communication and capacity in a 3d stacked reconfigurable cache hierarchy. In *HPCA*, 2009.
[14] K. Puttaswamy and G. H. Loh. Dynamic instruction schedulers in a 3-dimensional integration technology. In *GLSVLSI*, 2006.
[15] V. Rao et al. Tsv interposer fabrication for 3d ic packaging. In *Electronics Packaging Technology Conference*, 2009.
[16] K. Skadron et al. Temperature aware microarchitecture. In *ACM SIGARCH Computer Architecture News*, volume 31, pages 2–13. ACM, 2003.
[17] D. Tullsen. Simulation and modeling of a simultaneous multithreading processor. In *CMG*, 1996.
[18] Y. Watanabe et al. WiDGET: Wisconsin decoupled grid execution tiles. In *ISCA*, June 2010.
[19] D. H. Woo et al. An optimized 3d-stacked memory architecture by exploiting excessive, high-density tsv bandwidth. In *HPCA*, 2010.