# Modeling and Analysis of Dynamic Core Processors and their Application in the Mobile Marketspace

Kyle Daruwalla, David McNeil, and Ben Schmidt
*Rose-Hulman Institute of Technology*

***Abstract*—As both general purpose, desktop-grade processors and embedded processors mature, the distinguishing gap between them continues to diminish. Out of this narrowing market, a need for a versatile, low-power core emerges. Multicore embedded processors are just over the horizon. Dynamic core processors attempt harness these low-power cores to maximize both parallel throughput and minimize serial latency.**

## I. Introduction and Motivation

Mobile computing devices have grown to require processors that can support a dynamic workload. The typical serial tasks such as voice compression, speech transcoding (for communications), and image compression must perform well. However, newer mobile devices need to render complex graphics and run advanced background scheduling, all while maintaining serial performance and power. The obvious answer might be GPUs, but they are known to be power-hungry chips. Ideally, if most of the serial and parallel tasks could be performed on a single chip, the GPU would only need to be used to perform high thread level parallelism tasks like displaying graphics. A smaller GPU workload means a lower power GPU. Thus, power consumption can be kept minimal, while performance gains are still attained.

Dynamic core processors attempt to solve precisely these issues. By adding an interconnection network, some additional hardware, and control logic, the symmetric multicore embedded processors can be reconfigured into a modern, super-scaler, out-of-order processor. Thus, by identifying serial and parallel program sections, the processor can be dynamically changed to perform efficiently.

Obviously, creating such a processor, though possible, is not a trivial task. So, we attempt to analyze the theoretical performance gains of a dynamic core as described above. The following work will show whether a dynamic core processor performs better than a symmetric multicore processor and a modern, super-scalar, out-of-order processor. Furthermore, for a fixed length program, we determine how often the program must switch from serial to parallel processing before a dynamic core processor realizes performance gains. Finally, we determine the maximum number of cores in a dynamic processor before critical path length negatively impacts serial performance.

## II. Prior Work

The motivation for this paper came from the paper Amdahl's Law in the Multicore Ere [1]. Here Amdahl's law was extended to cover multiple multiprocessing hardware models based on a limited budget of hardware resources, or BCEs (Basic Core Equivalents). Based on both the percent of the workload that is parallelizable, the distribution of BCEss, and how performance of a single core scales, these give new speedup formulas to model system level hardware. The models used were Symmetric - same BCEs per core, Asymmetric - one large core and the rest small, and Dynamic - all BCEs switch between one large core and numerous small.

From there the curves were displayed for various total BCEs and fractions parallel, deriving results to consider for designing multicore systems. Among these results are that parallelism is still critical for achieving good speedups, increasing performance per core is globally efficient even if locally inefficient for symmetric and asymmetric. Furthermore, dynamic systems are better than asymmetric, which are better than symmetric. Based on this result for a dynamic system offering superior performance gain, we decided to investigate dynamic multiprocessing as an avenue to improve performance.

## III. Methodology

### A. Models

In order to simulate a dynamic core processor without actually implementing one, the team identified two basic models - a baseline symmetric core and a modern, out-of-order core. By combining multiple baseline cores into a single multiprocessor, we were able to create a parallel model (i.e. a processor biased towards parallel programs). The single modern, out-of-order core makes our serial model (i.e. a processor biased towards serial programs). Thus, the third model in our study, the dynamic model, is a combination of the serial and parallel models.

Based on this methodology, we needed a simulator capable of switching between CPU types. Previous work in this area had been done using the gem5 simulator. Capable of switching between CPU types, gem5 was an ideal choice.

### B. Tools

The gem5 simulator [2] is a combination of M5, a simulation framework with support for multiple ISAs and CPU models, and GEMS, a memory simulation system in two parts - Ruby and Opal. As a result, gem5 is a robust simulator with support for five ISAs, ARM, ALPHA, MIPS, Power, SPARC, and x86, and four CPU models:

- AtomicSimple is a minimal single IPC CPU model
- TimingSimple is similar to AtomicSimple but also simulates the timing of memory references,

- InOrder is a pipelined, in-order CPU
- O3 is a pipelined, out-of-order CPU model

gem5 has two modes of operation - system emulation (SE) mode and full system (FS) mode. System emulation mode does not model the OS or peripheral devices but solely simulates the specified benchmark. Full system mode, on the other hand, uses an actual OS kernel and mounts a Linux disk image. Essentially, gem5 full system mode is capable of booting a full OS and presenting the user with a Linux command prompt.

Initially, gem5's many high level customizations suggested that it would be an effective simulator for a dynamic processor model. However, while gem5 may boast many impressive features, we found that many of these features are not fully supported or difficult to configure.

### C. *Model Accuracy*

For simulation of a dynamic processor, we needed two types of benchmarks. One which would be representative of highly parallelized workloads and one reflecting serial execution. The parallel benchmark we used was SPLASH-2 [3]. SPLASH-2 is a suite of benchmarks intended to evaluate the performance of multiprocessor systems. For a serial benchmark suite, we chose to use MediaBench II [4]. A suite of compression and decompression multimedia algorithms. The programs selected from each benchmark were intended to accurately reflect typical mobile device usage. Figures 1 and 2 detail the exact benchmarks used.

### D. *Procedure*

When we initially began the project we planned on using gem5's SE mode for simplicity. However, due to the parallel nature of SPLASH-2 a multi-threading library is required. Because SE mode does not emulate a full operating system, there was no OS-level threading library such as pthreads. As a result, we finally decided to use full system mode. This provided us with access to Linux's full threading library allowing us to simulate the multi-threaded applications. Using full system mode had the additional benefit of providing extremely accurate results because the benchmarks are actually running in a Linux operating system, similar to our application space.

At this point, it was necessary to determine what CPU types and ISAs would be tested in order to simulate a dynamic processor. We initially planned on using either x86 or ARM as the underlying ISA. However, we soon found that only the AtomicSimple CPU model was supported by gem5 full system mode for these ISAs. gem5 provides much more complete support for the ALPHA ISA. ALPHA is a 64-bit RISC ISA representative of a simply architected general processor. Using this ISA as our building block we were able to develop benchmarks to represent a serial and a parallel machine. Our serial system used the O3 CPU model to simulate a modern out of order processor. Our parallel system was comprised of multiple TimingSimple CPU models. Originally, we had intended to use the InOrder model, however it was not available. Even though the TimingSimple CPU type was not a standard in-order pipeline, an array of the single cycle cores out performed the out-of-order processor on most parallel benchmarks. Thus, we felt the loss in accuracy was negligible.

In order to efficiently run tests, we developed scripts to boot a Linux kernel in FS mode and run through each of our benchmarks. At the completion of each benchmark the timing statistics would be written out to file. A script was then developed capable of parsing the output file and produce a CSV file containing the statistics for each benchmark. The Linux boot process became an advantage in our setup. Spanning 2.4 trillion instructions, we could be confident that all our caches would be warmed up after the boot process was complete. Furthermore, gem5's built-in stats-dump tool allowed us to run serial and parallel benchmarks one after the other without shutting the system down. This meant that the cache state before each benchmark was exactly the cache state at the end of the previous benchmark; thus, the initial cache miss latencies were embedded into our timing results.

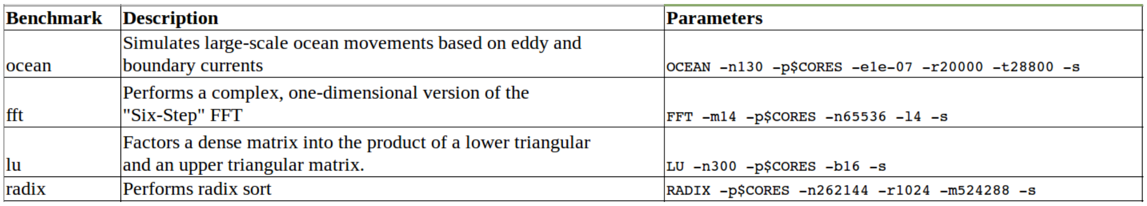
Once we had the timing results for the serial and parallel models, we synthesized the results together to get the timing performance for a dynamic core. Of course, there is some latency associated with switching from a serial configuration to a dynamic configuration. Though we did not implement a dynamic core, we did theorize potential methods for a datapath. In particular, we were drawn to the idea of using muxes to run external inputs into the execution units of several cores. Based on this, it seemed appropriate that an advanced interconnect network would be required. Most multicore and system on a chip (SoC) devices use a crossbar network, but the crossbar is hardly scalable. Instead, many bleeding-edge devices use network-on-chip (NoC) structures that are based off of crossbar interconnects, but optimized for scalability and power dissapation. Thus, we selected a switching latency of 0.991 ns based on research in the field of NoCs [5].

Furthermore, in order to study the problems surrounding a dynamic processor, we tackled two optimization issues. First, we chose a single serial program and a single parallel program to make up a single "chunk" of mixed code. We then look run time for 512 chunks put together. Then, the size of the serial and parallel portion in a chunk were doubled, so that each chunk is twice as large as it was before. However, we now only use 256 chunks in the test, so that the overall number of instructions remains constant. Repeating this process of doubling the size of a serial chunk and a parllel chunk while halving the number of chunks in the program, we are able to get results for the number of switches for maximum performance between serial and parallel portions within a fixed code segment. Secondly, we recognized that as the number of cores in the parallel model increases, the critical path of the serial model increases. Thus, the frequency should scale down for the serial model. Based on simple geometry (Pythagorean's theorem), we assume that the frequency will scale on the order of $1/\sqrt{n}$. Thus, we obtained results for the serial model with scaled frequencies to see if there is a drop-off in performance.

## IV. Results

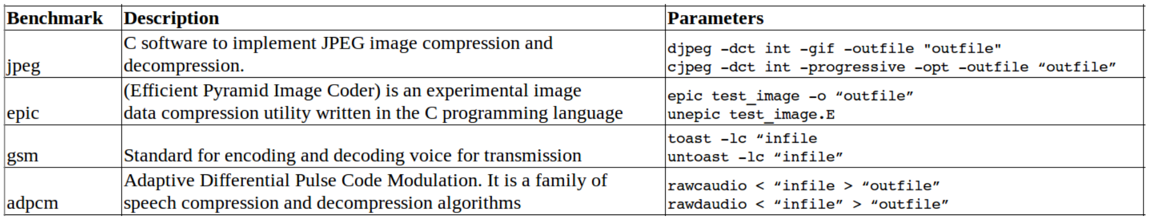
### A. *Model Accuracy*

First, we needed to verify that our serial and parallel models were accurate. As expected, the serial model out performed



| Benchmark | Description | Parameters |
|---|---|---|
| ocean | Simulates large-scale ocean movements based on eddy and boundary currents | `OCEAN -n130 -p$CORES -e1e-07 -r20000 -t28800 -s` |
| fft | Performs a complex, one-dimensional version of the "Six-Step" FFT | `FFT -m14 -p$CORES -n65536 -l4 -s` |
| lu | Factors a dense matrix into the product of a lower triangular and an upper triangular matrix. | `LU -n300 -p$CORES -b16 -s` |
| radix | Performs radix sort | `RADIX -p$CORES -n262144 -r1024 -m524288 -s` |

**Fig. 1:** *The specific SPLASH-2 benchmarks used*



| Benchmark | Description | Parameters |
|---|---|---|
| jpeg | C software to implement JPEG image compression and decompression. | `djpeg -dct int -gif -outfile "outfile"`<br>`cjpeg -dct int -progressive -opt -outfile “outfile”` |
| epic | (Efficient Pyramid Image Coder) is an experimental image data compression utility written in the C programming language | `epic test_image -o “outfile”`<br>`unepic test_image.E` |
| gsm | Standard for encoding and decoding voice for transmission | `toast -lc “infile`<br>`untoast -lc “infile”` |
| adpcm | Adaptive Differential Pulse Code Modulation. It is a family of speech compression and decompression algorithms | `rawcaudio < “infile > “outfile”`<br>`rawdaudio < “infile” > “outfile”` |

**Fig. 2:** *The specific MediaBench II benchmarks used*

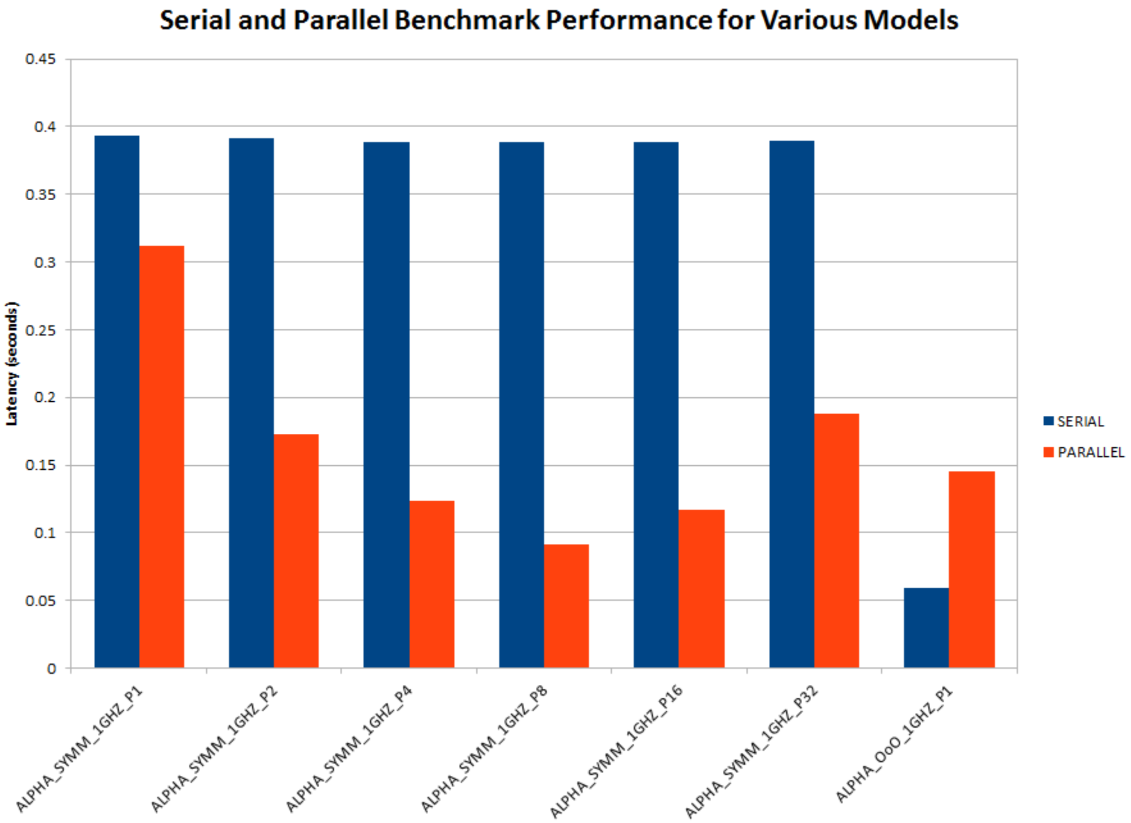


**Fig. 3:** *The serial and parallel models perform appropriately and accurately for different benchmarks (P16 refers to 16 cores)*

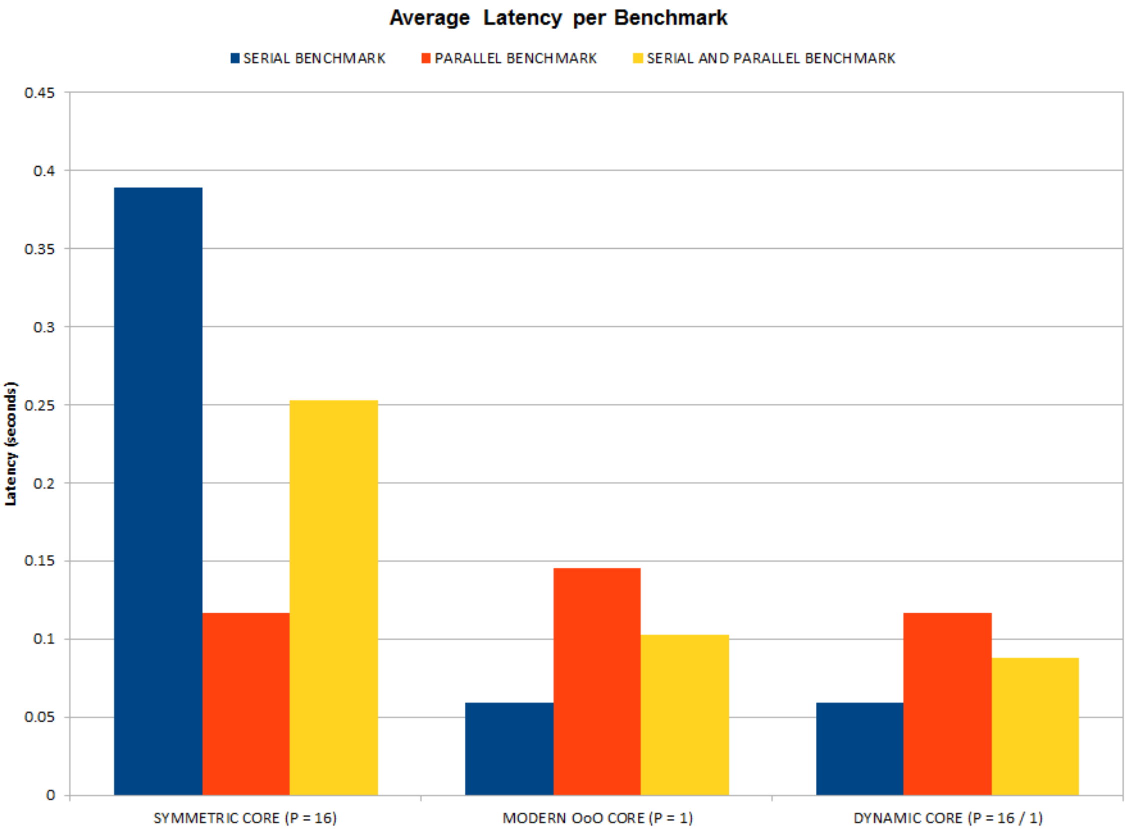


**Fig. 4:** *A dynamic core performs better than a 16 symmetric multicore machine and a single modern out-of-order machine*

the parallel model at serial tasks. Additionally, the parallel model performed the same for a serial task regardless of the number of cores; this is expected, since only a single core can effectively be used. Finally, the parallel model performed better that the serial model for parallel programs. However, it is worth noting that as the number of cores increased passed eight, the parallel performance began to degrade. At 32 cores, the performance was even worse than the serial model. Even so, we feel that these models are accurate for our theoretical study. See Figure 3 for these results.

### *B. Is Dynamic Better?*

Ultimately, we wanted to answer the question “Is Dynamic Better?”. At this stage, “better” is simply evaluated in terms of performance (we will go into more detail later). As can be seen in Figure 4, a dynamic core performs better in terms of average latency over the dynamic benchmark. Based on these results, the dynamic core has about a 4x improvement over a single baseline core machine (a basline core being the cores used to make up the parallel model), a 2.9x improvement over a sixteen core symmetric multicore machine, and 1.2x improvement over a modern, out-of-order processor. Note that we assumed the TimingSimple model in gem5 was sufficient to be used as our baseline core. This conclusion is corroborated by the results in Figure 3; yet, if we had used a RISC pipelined architecture instead of a single cycle processor, the parallel model performance would have been even better (note how small the difference is between the parallel model and the serial model for the parallel benchmark in Figure 4). So, if anything, our dynamic core would have even greater performance gains.

However, as we briefly mentioned before, “better” is more than just performance. Especially on a mobile device, “better” is determined by efficiency. While we did not quantitatively determine the power consumption, we can qualitatively say a dynamic core will be more power efficient by nature. Since the symmetric multicore is made up of small, low power cores, and these same cores make up the basis for a dynamic core, a dynamic core is bound to be power efficient. There is, of course, overhead associated with a dynamic core. But this overhead should be no more than the overhead involved in a modern, out-of-order processor. We can analytically estimate this power overhead. Based on research at MIT, the maximum total power for a flip-flop or latch in a high performance system is 350 uW [6]. Given that we are simulating an eight issue-width modern, out-of-order processor, we can expect about eight functional units (functional units scale with issue width). Since each function unit requires two inputs and one output (32-bits each) to be muxed, each functional unit will have $(3muxes)((32latches) + (32flip - flops)) =$

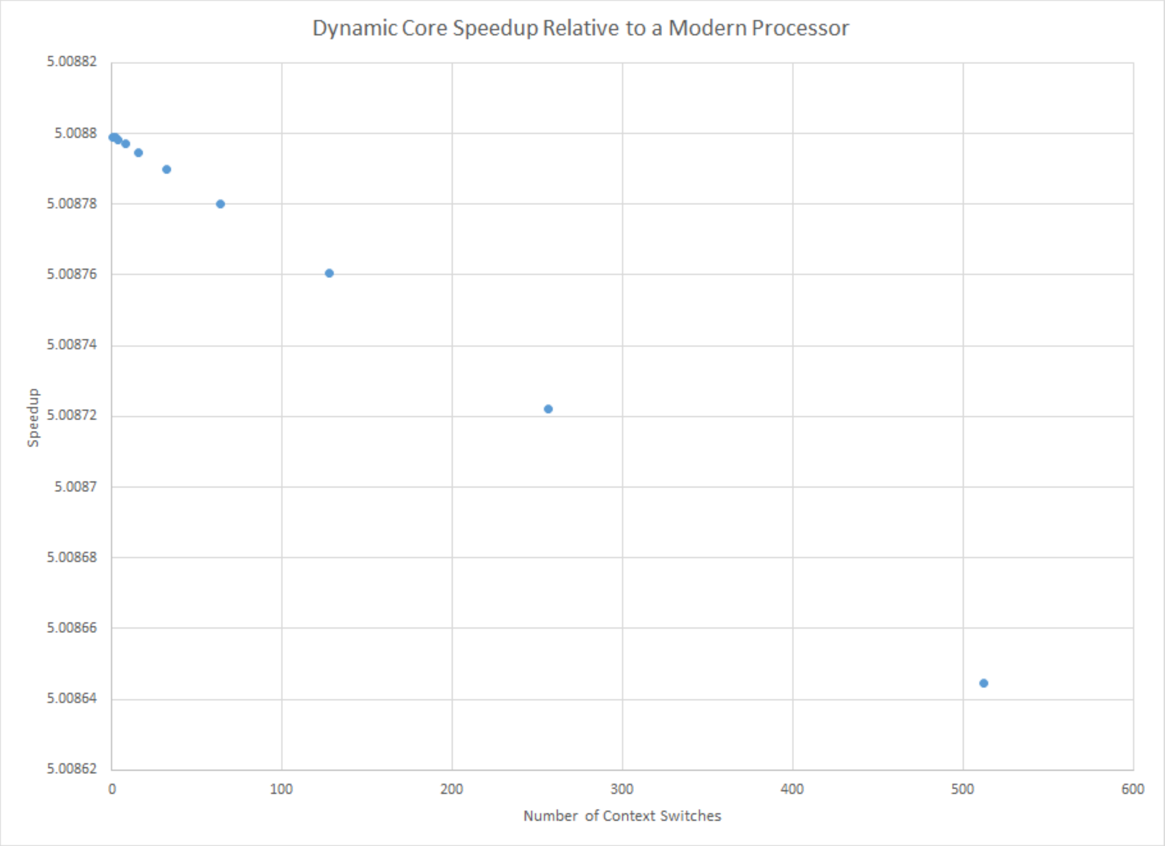


**Fig. 5:** *Fine granurality switching degrades performance gains*

$192 latches/flip-flops$. So, the average power overhead is $0.350(8)(192) = 537.6$ mW, which is minimal compared to the power consumption of a modern, out-of-order processor. So, speculatively (with some evidence), we can say that a dynamic core should consume less power than a modern, out-of-order processor, while still performing better.

### C. Peak Number of Switches

When discussing a processor that performs context switching, it is useful to mention granurality. In other words, how often should a processor be switching between serial and parallel mode. As Figure 5 shows, the finer grained the context switching, the lower the performance gains. Do not let the scale of the graph be misleading, the problem has purposefully been scaled up. So, our smallest building block between a context switch is a full program long (i.e. a full FFT or voice compression algorithm). Thus, the degradation is not as drastic numerically. However, logically, the trend should hold as the granularity becomes finer. Since the only factor degrading the performance is a linear constant for the interconnect latency, it follows that very fine grained switching will lower performance gains visibly.

### D. Frequency Scaling

Though increasing the cores in a dynamic processor will result in performance gains due the to the parallel mode running faster, the serial portion will begin to suffer. A large number of cores means a greater area, which mean a longer critical path for the serial model. Based of off Pythagorean's theorem, we hypothesized that as the number of cores increased, the frequency would scale on the order of $1/\sqrt{n}$. After re-running our original tests, we determined the new dynamic core latencies with a scaled frequency serial model. The speedup relative to a 1 GHz modern, out-of-order core can be seen in Figure 6. There is a peak performance gain at four cores, before the speedup decays. So, while a dynamic core is better in an ideal sense, when practical constraints are applied to it, performance begins to degrade.

### E. Practicality

In addition to our theoretical models, we chose to compare our dynamic core to modern, out-of-order, dual core processor, which is typical for the average device today. The speedup was 0.75x, so a decrease in performance. One arguement in favor of a dynamic core might be that a dual core, eigth issue width, out-of-order processor is equivalent to a single sixteen issue width out-of-order processor when we convert it to a serial

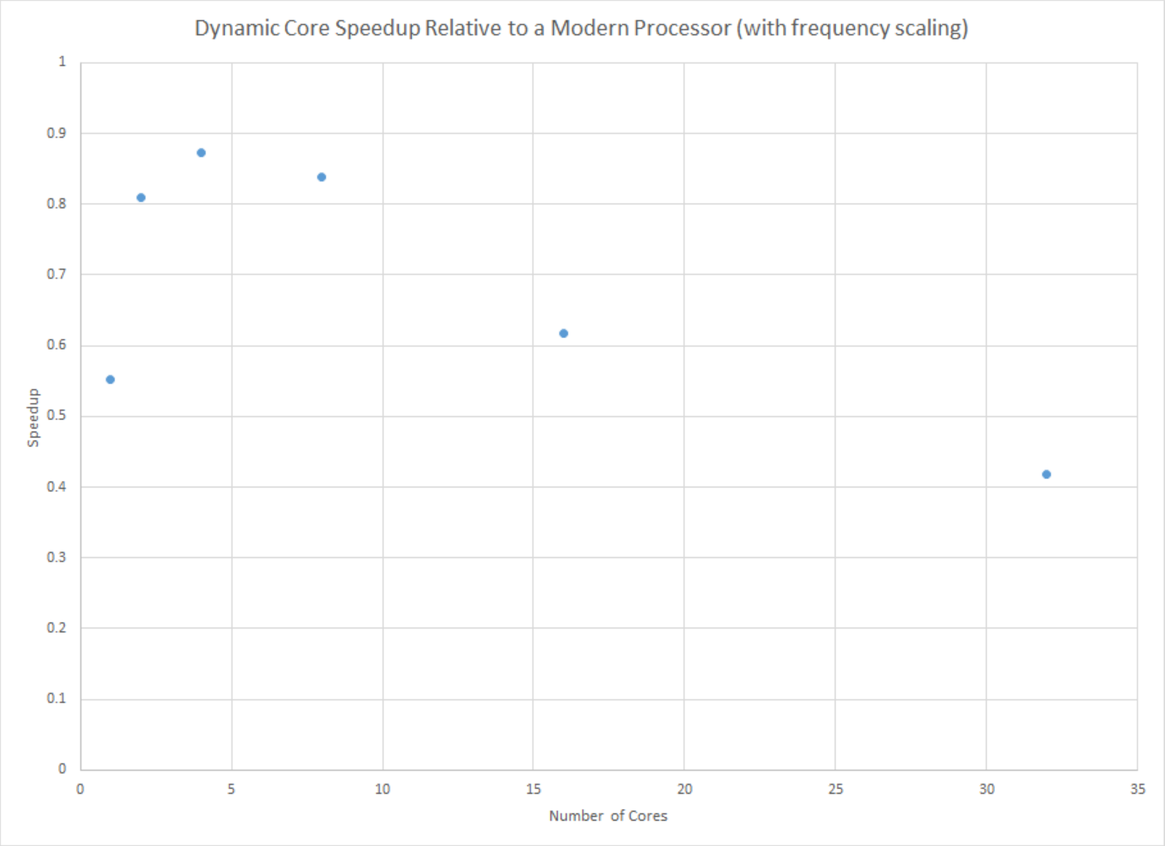


**Fig. 6:** *Increasing the number of cores scales down the serial model frequency, resulting in a peak performance at four cores*

model. We considered this, but noticed little improvement between a serial model using an eight issue width core and a sixteen issue width core.

## V. Conclusion and Future Work

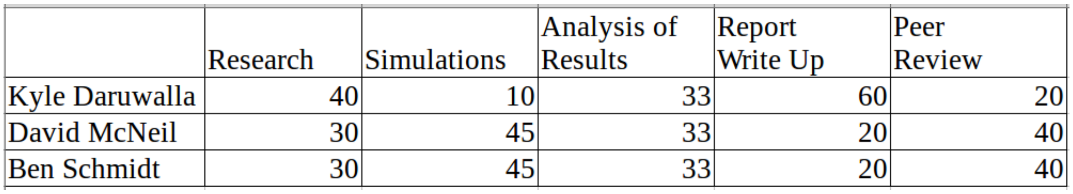
Overall, the results of the study were promising. A dynamic core performs better than both a symmetric multicore machine, as well as a modern, out-of-order processor. We identified an approximate power overhead for a dynamic core; we argue that this overhead is minimal compared to the total power consumption, and that a dynamic core receives 1.2x speedup with a small loss in power efficiency. Furthermore, we identified that fine grained context switching between serial and parallel workloads degrades performance on a dynamic core. Finally, the study noted that there is peak performance gain for a dynamic core as the number of baseline cores is scaled. This was due to an increased critical path for the serial model.

Even though the dynamic core did not perform better than a dual core modern, out-of-order processor, we were able to identify areas for improvement. First, creative critical path routing will significantly help the performance of a dynamic core. Secondly, coarse grained context switching will maximize performance. As always, it was clear that Amdahl's Law still holds true, and both serial and parallel performance need to be patiently maximized to gain better overall performance.

The critical path issue lends itself nicely to 3D stacking technologies. Take, for example, a baseline core that is serving as a functional unit for the serial model. This core will receive data from another baseline core's register file. Using 3D stacking, the two cores could be placed on top of each other, so that the path from the register file of one core to the execution unit of another is small.

In addition to analyzing the performance gains of 3D stacking technologies, the study would be best served by a reanalysis of the serial and parallel models. For our theoretical study, the ALPHA ISA was sufficient, but a more accurate serial model would be an out-of-order x86 core, and a more accurate parallel model would be a multicore ARMv7 machine.

Keeping this future work in mind, there is still plenty of room for the dynamic core model to grow. Theoretical research such as ours needs to be done, as well as implementation research to get more accurate latency times. Dynamic cores alone will not help mobile device manufacturers, but a combination of dynamic cores with other technologies could prove to be an effective speedup with minimal loss in power efficiency.



| | Research | Simulations | Analysis of Results | Report Write Up | Peer Review |
|---|---|---|---|---|---|
| Kyle Daruwalla | 40 | 10 | 33 | 60 | 20 |
| David McNeil | 30 | 45 | 33 | 20 | 40 |
| Ben Schmidt | 30 | 45 | 33 | 20 | 40 |

**Fig. 7:** *Contributions of individual team members*

## VI. Statement of Work

See Figure 7
Signed by: Kyle Daruwalla, David McNeil, and Ben Schmidt

## Acknowledgment

The authors would like to thank Dr. Daniel Chang for providing them with basic knowledge to conduct the research.

## References

[1] Mark D. Hill, Michael R. Marty. “Amdahls Law in the Multicore Era” [Online] Available: http://moodle.rose-hulman.edu/pluginfile.php/245005/mod_resource/content/0/Amdahl_Multicore%20%28Hill%29.pdf

[2] N. Binkert, B. Beckmann, G. Black, S. Reinhardt, A. Saidi, A. Basu, J Hestness, D. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. Hill, D. Wood. “The gem5 Simulator” [Online] Available: http://research.cs.wisc.edu/multifacet/papers/can11_gem5.pdf

[3] Computer Architecture and Parallel Systems Library. *Modified Splash2*. Newark, DE. CAPSL, 2007. Available: http://www.capsl.udel.edu/splash/index.html

[4] University of California - LA. *MediaBench II*. Los Angeles, CA. Available: http://euler.slu.edu/~fritts/mediabench/

[5] Sung-Joon Lee, Jaeha Kim. “A 256-Radix Crossbar Switch Using Mux-Matrix-Mux Folded-Clos Topology” [Online] Available: http://www.jsts.org/html/journal/journal_files/2014/12/Year2014Volume14_06_10.pdf

[6] Vladimir Stojanovic, Vojin Oklobdzija, Raminder Bajwa. “Comparative Analysis of Latches and Flip-Flops for High-Performance Systems” [Online] Available: http://www.rle.mit.edu/isg/documents/Stojanovic_ICCD98.pdf