Dynamic Core Processors

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.**

***Keywords*—IEEEtran, journal, LaTeX, paper, template.**

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 TLP 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 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.

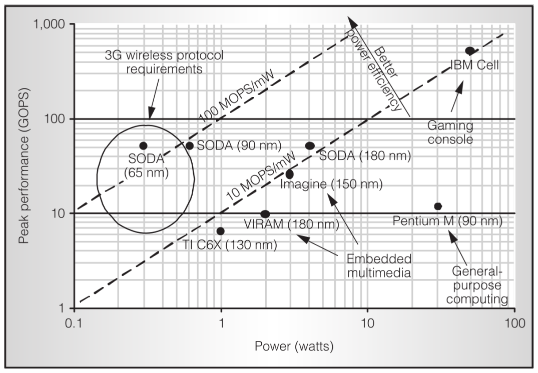

Fig. 1: *THIS IS JUST AN EXAMPLE IMAGE FROM ANOTHER PAPER Performance and power requirements of 3G wireless. Theoretical performance and power statistics of SODA, other DSPs, and general-purpose processors [1].*

testing images Fig. 1

II. Prior Work

Summarize relevant related work with appropriate citations.

III. Methodology

Describe your methodology in implementing your project. If a simulator is used, cite it accordingly and detail what changes you made. If a structure was implemented, give details as to what was done. If an algorithm was studied, give details as to its operation. Block diagrams, flow charts, and detailed examples can be very helpful.

The gem5 [1] simulator was used to simulate the performance of a dynamic processor. gem5 is a combination of the M5, a simulation framework with support for multiple ISAs and CPU models, and GEMS, a memory simulation system. 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 potential modes of operation system mode and full system mode. System modes 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. With gem5 great degree of high level customizations we concluded that it would

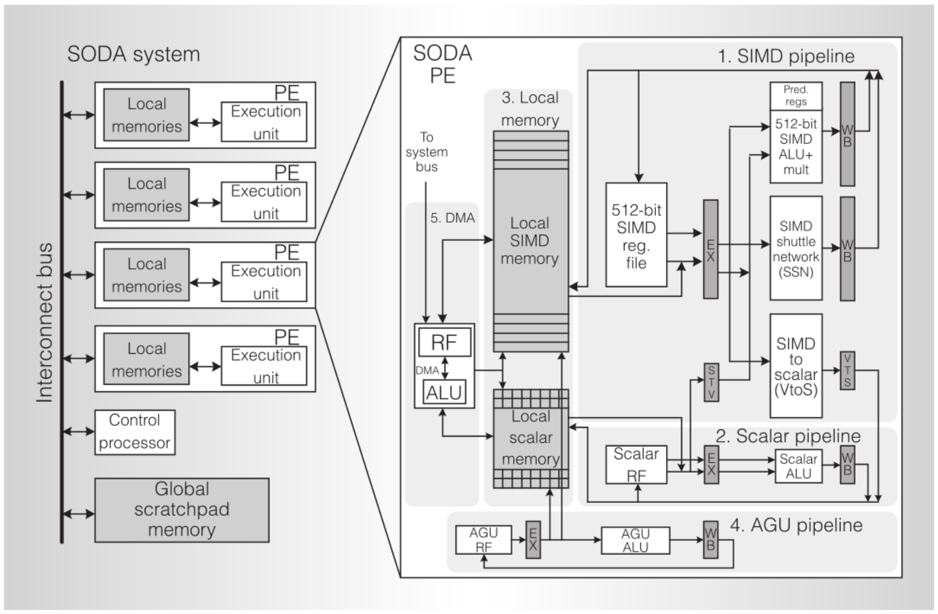

Fig. 2: *THIS IS JUST AN EXAMPLE IMAGE FROM ANOTHER PAPER The SODA Architecture [1]*

Benchmark	Description	Parameters
OCEAN	placeholder	placeholder
FFT	placeholder	placeholder
LU	placeholder	placeholder
RADIX	placeholder	placeholder

TABLE I: The SPLASH-2 benchmarks used

Benchmark	Description	Parameters
jpeg	placeholder	placeholder
epic	placeholder	placeholder
gsm	placeholder	placeholder
adpcm	placeholder	placeholder

TABLE II: The MediaBench II benchmarks used

be an effective simulator for simulating a dynamic processor. However, while gem5 boasts many impressive features, we found that many of these features are not fully supported or difficult to configure.

For simulation of a dynamic processor, we needed benchmarks which would be representative of highly parallelized workloads and a benchmark reflecting serial execution. The parallel benchmark we used was SPLASH-2 [2]. SPLASH-2 is a suite of numerous parallel benchmarks intended to evaluate the performance of multiprocessor systems. For a serial benchmark suite we choose to use MediaBench II [3] which is a suite of numerous compression and decompression multimedia algorithms. Tables **??** and **??** detail the exact benchmarks used.

When we initially began the project we planned on using gem5 system mode emulation for simplicity. However, due to the parallel nature of SPLASH-2 a multi-threading library is required. Because system mode does not emulate a full operating system the threaded nature of these applications was impossible to simulate as a result we finally decided to use full system mode. This provided us with access to Linux full threading library allowing us to simulate the multi-threaded applications. Using full system mode has the additional benefit of providing extremely accurate results because the benchmarks are actually running in a Linux operating system.

gem5 provides an API for writing the current timing statistics out to a file. We then developed scripts to run the benchmarks and generate timing information about each benchmark. The simulator was then run using different configurations of CPU type and number of cores.

IV. Results

Provide and justify any quantitative results, preferably using graphs and/or tables.

V. Conclusion and Future Work

Summarize your findings, identify remaining open problems or issues, and suggest the next steps for this project.

VI. Statement of Work

The project report must also include a statem ent of work that identifies the contributions of each individual on the team.

The statement of work must reflect a team consensus and must be signed by all team members. I recommend that you structure this statement as a table with a row for each project m ilestone, a column for each team member, and the percentage contribution of each team member to each milestone in the entries in the table. Please do not give a vanilla 50/50 in every column.

Acknowledgment

The authors would like to thank...

References

[1] 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," Available: http://research.cs.wisc.edu/multifacet/papers/can11_gem5.pdf

[2] Not sure how to site this. Available: http://www.capsl.udel.edu/splash/index.html

[3] Not sure how to site this. Available: http://euler.slu.edu/~fritts/mediabench/