Project 2 Report: Evaluation of Asymmetrical Architecture

Authors: Ujjwal Sai.K

Haoxuan Zhang

INTRODUCTION:

As the Chip Multiprocessors started to flourish, different architectures were looked into for performance benefits, at least under special conditions. One such advancement is Asymmetric computer architecture or Asymmetric multiprocessing in which the processing power of the cores is varied and the cores are not treated equally. These systems gain performance by incorporating cores with different processing strengths or cores that have specialized processing capabilities to handle particular tasks. For instance, only one CPU might be able to execute operating system code and some other CPU might only be allowed to perform I/O operations. This mainly helps when there is a lack of parallelism in the data. We intend to show that for a specific set of programs, heterogeneous computing could provide a speedup over symmetric multiprocessors through a set of experiments on the GEM5 architectural simulator with PARSEC benchmarks. Roughly, we followed the following steps.

Throughout the experiment, we utilize a concept called the "criticality" defined in [1]. This notion of the criticality of a thread of execution combines, both how much time a thread is performing useful work and how many co-running threads are waiting. This criticality can be calculated periodically, called "criticality stacks", for different time slices to provide easy visualization of processor utilization throughout the execution.

- ➔ The criticality of a thread 'j' is calculated by:
  - ◆ $C_j = \sum_{i=0}^{N-1} \left\{ \frac{ti}{ri} \right\} \; if\, j \; \varepsilon \; R_i$ and '0' otherwise
    - Where t_i is the duration of interval i, r_i is the number of running threads in that interval, and R_i is the set containing the thread IDs of the running threads over N such intervals.

Steps for showing performance improvement

- Profiling
  - Modify the statistics in Gem5 to figure out criticality during each period of time
  - Run Simple CPU and dump the statistics periodically (according to the period we set)
  - Create a table with criticality for each thread for all the execution periods. Figure out the most critical thread for each period.

- Simulate the cores (CPI)
  - Fix the full system o3 CPU error
  - Run the benchmark for a specific number of Instructions. Here the smaller core for comparison would be a timing simple CPU with 1 thread and the larger CPU would be an Out-of-Order CPU with the same threads.

- Calculate runtime by using CPI frequency and No: instruction for Critical Thread
  - For each period defined above, we figure out the most critical thread. We use the CPI and frequency(or Instruction per second) to calculate the runtime for each period.
  - Compare the runtimes from this step (Accelerated CPU) and the one from step 1 (Base CPU) to figure out runtime improvement!

Simulated System Configuration:

- ➢ Profiling part: For the multicore system that we intend to accelerate with asymmetrical architecture, we used 8 cores with Timing Simple CPU model. The following table shows the cache size for our system:

L1 data	64 KB
L1 Instruction	32 KB
L2	2MB
L3	16 MB

- ➢ The coherence protocol used here is the MOESI_hammer protocol. The testbench used is blackscholes and the command to run the benchmark in .rcS file is as follows:

```
./blackscholes 8 /parsec/install/inputs/blackscholes/in_4K.txt
/parsec/install/inputs/blackscholes/prices.txt
```

- ➢ The system clock and CPU clock are 1GHz and 2GHz respectively.

- ➢ Speed up simulation part: We simulated out of order core and in-order timing simple CPU with only one core running blackscholes benchmark with the same size as the profiling part. The frequency and cache configurations are all the same.

- The assumed asymmetrical architecture: We assume an architecture that has totally 8 cores with one of them to be a larger out-of-order core with issue width, execution width and commit width all to be 8, while all other cores remain to be the in-order core using Timing Simple CPU model. We would find the critical thread in symmetrical architecture for every period of time and assume that the scheduler could be smart enough to schedule that thread on the larger core in the asymmetrical architecture.

Calculation Method:

We use the following steps to evaluate the performance of the asymmetrical architecture mentioned in the previous section:

- Do the simulation for a single-core CPU in the DerivO3CPU model running blackschole benchmark to acquire the instruction per second of the larger core: IPS_1 .

- Run the simulation for a single-core CPU in the TimingSimpleCPU model and acquire the instruction per second of the small cores: IPS_2 .

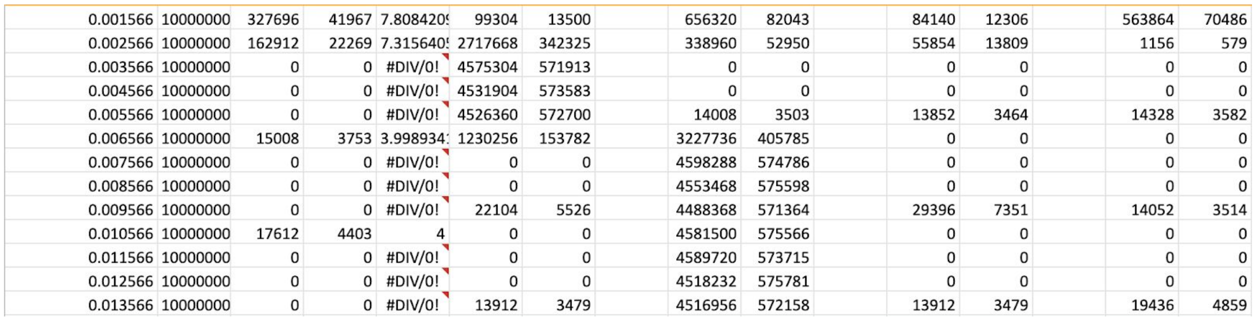
- Run the simulation with 8 TimingSimpleCPU cores and get the Criticality for each core of every fixed period of time.

➢ For each time slot, we select the core having the largest Criticality and the second-largest Criticality. The instruction count of the core having the largest Criticality is I_1 , while the instruction count of the core having the second-largest criticality is I_2 . Let $T_1 = I_1 \div IPS_1$ and $T_2 = I_2 \div IPS_2$. The time asymmetrical architecture system will spend in the same amount workload would be max$\{ T_1, T_2 \}$. The simulation and evaluation result is shown in the next section.

RESULTS:

After our series of tests, we used a python script, as explained above, to gather all the criticalities for each time slice for each of the 8 cores. In addition, we also used TimingSimpleCPU and O3CPU to find the exact speedup offered by the latter processor with one core on the PARSEC “blackscholes_simsmall” benchmark. We use these results in union with the above criticalities to paint a picture of how a stronger processor could provide computational performance benefit over a symmetric architecture.

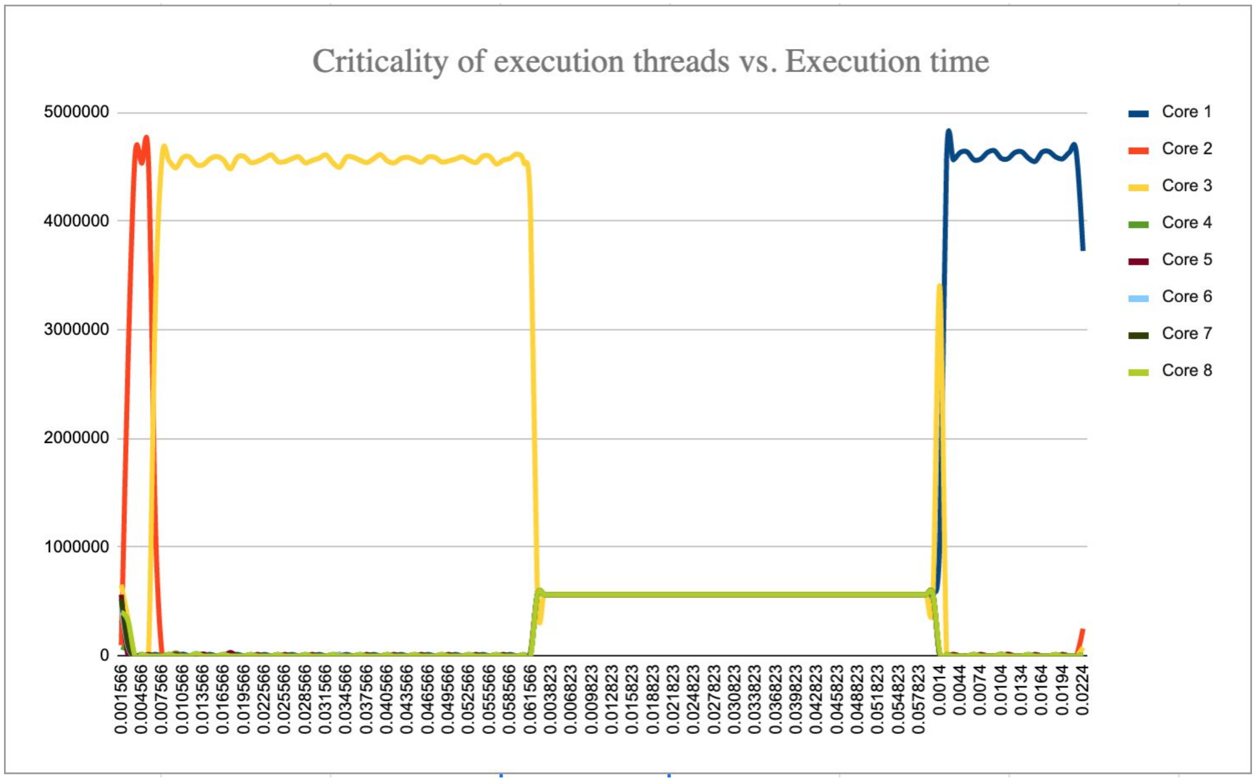
➢ A sample of our data is given below:



0.001566	10000000	327696	41967	7.8084209	99304	13500		656320	82043		84140	12306		563864	70486
0.002566	10000000	162912	22269	7.3156405	2717668	342325		338960	52950		55854	13809		1156	579
0.003566	10000000	0	0	#DIV/0!	4575304	571913		0	0		0	0		0	0
0.004566	10000000	0	0	#DIV/0!	4531904	573583		0	0		0	0		0	0
0.005566	10000000	0	0	#DIV/0!	4526360	572700		14008	3503		13852	3464		14328	3582
0.006566	10000000	15008	3753	3.9989341	1230256	153782		3227736	405785		0	0		0	0
0.007566	10000000	0	0	#DIV/0!	0	0		4598288	574786		0	0		0	0
0.008566	10000000	0	0	#DIV/0!	0	0		4553468	575598		0	0		0	0
0.009566	10000000	0	0	#DIV/0!	22104	5526		4488368	571364		29396	7351		14052	3514
0.010566	10000000	17612	4403	4	0	0		4581500	575566		0	0		0	0
0.011566	10000000	0	0	#DIV/0!	0	0		4589720	573715		0	0		0	0
0.012566	10000000	0	0	#DIV/0!	0	0		4518232	575781		0	0		0	0
0.013566	10000000	0	0	#DIV/0!	13912	3479		4516956	572158		13912	3479		19436	4859

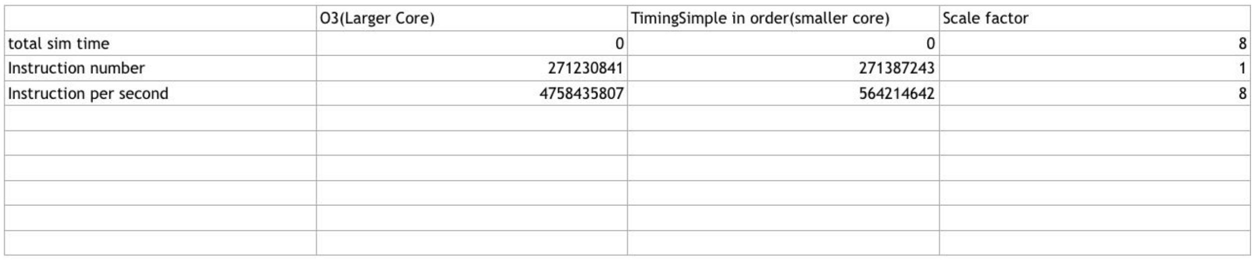
- The first column is the time slice, the second one is simulation ticks. From that point on, every set of 2 columns, with a 3rd gap column, depicts the criticality and number of instructions of 8 different cores from one to eight.

➢ Plotting this entire data for the entire execution period of ROI:



- As we can see from the graph, the criticality of certain cores is really high for a considerable portion of the execution showing/proving that a powerful core could take advantage of the excessive criticality and provide performance benefits in those time periods. At some point in the execution, say with a lot of parallelism, all the threads share the same criticality and are treated equally. It is also interesting to note that the time slices change mid execution before entering the “parallel phase”, maybe because of some kind of context switch.

➢ To find the Speedup among our different cores:

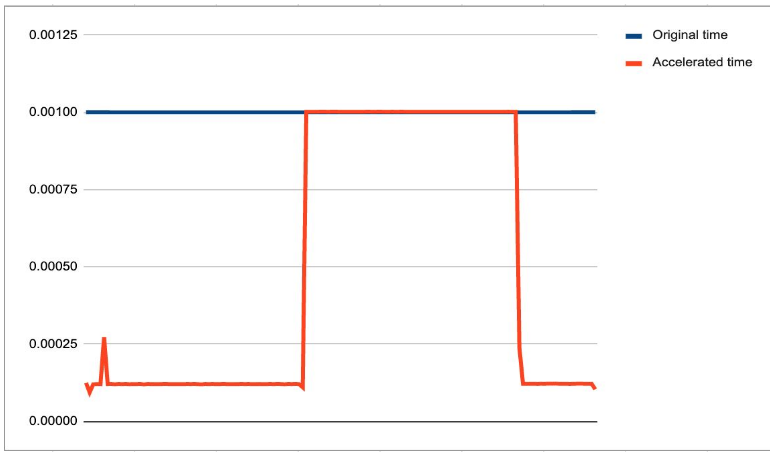


	O3(Larger Core)	TimingSimple in order(smaller core)	Scale factor
total sim time	0	0	8
Instruction number	271230841	271387243	1
Instruction per second	4758435807	564214642	8

- As we can see from the above table, the IPS for the O3CPU, larger CPU, is ~ 8 times that of the, smaller, TimingSimpleCPU. This gives us a scaling factor of 8 to check for our performance improvement!

➢ Plotting the “Accelerated time” for the O3CPU and “Original time” for the TimingSimpleCPU:

Execution time for time slice vs. Execution time slice



- From the graph, at least for half of the total execution time, the larger CPU has provided better execution time. The only portion of execution when the execution times are comparable to the symmetric simple core is when say there is high parallelism in a portion of code.
- To corroborate our point, we can also see that two graphs plotted above share similarities in that when high criticality is observed like at the two extremes of execution, we also observe great speedup in execution. Conversely, when the criticality among all the threads is the same, the in-order core and out-of-order larger core have comparable execution times.

CONCLUSION:

Therefore, we conclude that there is promise in asymmetric architecture. It has both pros and cons and it is up to the user to select the appropriate programs to utilize this on so that we can extract these performance benefits. Therefore, Programs with comparatively less parallelism should favor this kind of architecture. Further research into various optimization for this architecture is sure to help future interests.

REFERENCES:

1. Du Bois, Kristof, et al. "Criticality stacks: Identifying critical threads in parallel programs using synchronization behavior." Proceedings of the 40th annual international symposium on computer architecture. 2013.