**Shared-Memory-Consistency-Models**

1. **Problem**

The paper concerns with memory consistency and concurrency efficiency, which is a necessary topic resulted from the concurrent programs. No matter the program is executed on a uni-processor or multi-processor machine, the instructions of the program can be re-ordered either by the hardware or the compiler. Thus, the outcomes of the memory operations might differ from that of the sequential execution.

**B. Importance**

Concurrency program and multi-core processing is ubiquitous nowadays. There often exists trade-off between efficiency and memory consistency. For ensuring a certain level of memory consistency, not only the hardware but also the software design would be involved. In a architecture with cache structures, cache coherence is important to guarantee correctness of the programs.

**C. Solution**

The authors formulate their memory-consistent models in both system and programmer center perspective. They propose a read and write instructions order relaxation method that can mostly ensure the atomicity requirement. Relaxed reordering models fall into two categories, one distinguishes instructions by type and impose ordering constraints, another one explicit impose memory operations’ order. The model classifies synchronous instructions into *acquire* and *release* types, which requests and grants permission to memory locations. The release consistency has three models that use memory barriers, which maintain memory operations order before or after them. A programming-centric model defines synchronous operations as any operations that form a race with other operations in any sequentially consistent execution, while all others are data operations. To ensure the program correctness, the compiler must differentiate synchronous and data operations. Compiler-led program instructions reordering provide more flexibility and hence better performance than the hardware based optimization, as indicated by the paper. But it increases the program complexity by exposing the programmers to more low-level optimization of the model. The paper also summarizes the commercial models that adopts the memory relaxation model but doesn’t provide any experiment statistics to evaluate the

Performance quantitatively.

**D. Comments**

The paper uses diagram to clearly explain the logic and efficacy of the system- and program-centric memory relaxation model. They demonstrate how memory inconsistency comes into play in different scenarios. The authors also introduce the commercial use of each type of models. One thing the authors could improve is to conduct experiments regarding these optimization and provide quantitative details.

**Pioneering Chiplet Technology and Design for the AMD EPYC and Ryzen Processor Families**

1. **Problem**

There has been a growing demand for computation power whilst the increase rate of transistor density on microchips have slowed down due to the heightened cost of producing chips. Since there is a practical ceiling on how large silicon die can be manufactured, producing large chip won’t suffice the demand either.

**B. Importance**

The prevalence of Machine Learning-related models have incurred enormous amount of training power. Due to the multiplicity of processor product designs, the cost of semiconductor manufacturing and the limitation of the silicon die size, the naive solutions of sufficing the computation power demand have been ruled out.

**C. Solution**

the authors present two use cases of the multi-chiplet deisgn based on disintegration, which has been existent in pre-Moore’s era.The first use case is AMD EPYC™ processor. EPYC adopts 4 identical chips of which the cost is 0.59 of the monolithic chip. Individual chiplet may be harvested and repurposed into multi-core chiplets of different designs even with some defects. Multi-chiplets incur some inter-chiplet latency. This is largely due to the remote memory request to DDRs on other chiplets. Regarding the 2nd generation of AMD EPYC, they use a dual-chip design with one chip utilizing the cutting-edge 7nm die and the other a mature 12nm die. They have measured that fully transforming a 14nm chip to 7nm will be much less cost-effective. To implement the multi-chiplet design, AMD has to deal with several challenges such as packaging with silicon interposers with latency and power efficiency requirements and satisfying different bump pitches to connect the chiplets to the package substrates. The 2nd Gen. AMD EPYC also requires IFOP hop for every memory request, providing much more uniform memory access latency. Finally, the authors use charts to evaluate the cost and the performance of the EPYC design. The gap of die cost between monolithic and multi-chiplet design increases super linearly with the number of core. A second use case is 2nd Gen. AMD Ryzen™, which consists of two CCDs and a “client IOD” constructed with the silicon design of 2nd Gen. AMD EPYC™. AMD Ryzen™ inherit the multi-chiplet benefits of AMD EPYC. As the parametric variations could lead to performance difference between the cores, Ryzen measures the performance of core complexes to help allocate the threads to the fastest core.

**C. Comments**

The authors have done a great job showing the chip design in diagrams and explaining the multi-chiplet rationale. They also present some simplistic quantitative cost comparison between multi-chiplet and monolithic chiplet design and performance comparison across multi-chiplet generations. But it could be also helpful if they present performance comparison between multi-chiplet and monolithic chiplet designs in terms of latency and throughput.