Report on Lab 2

Ayushi Agarwal
2018ANZ8503
ayushi.agarwal@cse.iitd.ac.in

1 Introduction

The goal of this report is to study the performance of SPEC CPU 2017 Benchmarks on different kinds of multiprocessor configurations. The extent of processor resource sharing varies among Chip Multiprocessing(CMP), Symmetric Multiprocessing(SMP) and Simultaneous Multi-threading(SMT). Hence, it has a direct impact on the performance of the workloads running on these systems. We report how the performance of these benchmarks are affected by the extent of this resource sharing and also analyzed how the performance of a particular benchmark on these systems change with the kind of workload that we are running on them.

2 Method

We have studied the different benchmark performances on three types of multiprocessor architectures which are:

- **Chip Multiprocessor (CMP)** : It is a multi-core architecture design (mostly a variant of SMP or Symmetric Multiprocessor) in which many CPU cores are present on the same monolithic integrated circuit. Because of Moore's Law, the on-chip transistor density has increased significantly. Because of the saturating single core CPU performance, the trends have moved toward multi-core architectures. These usually have higher area and power. All the cores share a global memory and the memory could be organized in the following two ways:
  - *UMA* : In these systems, the global memory is present as one unified block. All the cores take uniform time to access a particular memory location. But this only has a single socket (all cores on one chip).
  - *NUMA* : In these systems, the global memory is divided into banks that are locally placed with **different groups of cores**. Hence, the time to access a memory location is non-uniform across the **cores**, although the time required to access any of the main memory banks is similar (that is why CMP's are also variants of SMP). All NUMA nodes are on one socket(or chip) divided among the cores since it is a CMP.
- **Symmetric Multiprocessor (SMP)** : It is a type of multi-processor architecture in which two or more identical processors (*these can be separate chips (sockets) or multiple cores on the same chip*) are connected together by high-bandwidth networks. All the processors are managed by one single operating system. All the processors share a global memory. All the processors have symmetric or equal access times to the memory or to a particular memory bank.
  - *NUMA* : In these systems, usually the global memory is divided into banks that are locally placed with **different processors** or CMP's. Hence, the time to access a memory location is non-uniform across processors. (All NUMA nodes on one socket per chip). Although there can be multiple-socket SMP's with UMA memory organization as well.
  - *Different Sockets*: In this SMP configuration, we have two or more sockets or CMP chips connected via high bandwidth off-chip network. Within the individual Sockets or across the sockets (as discussed in the above point) we can have NUMA and UMA memory organizations.
- **Simultaneous Multi-threading (SMT)** : Today's Intel/AMD machines use Simultaneous Multi-threading (Intel's Hyper-threading) to exploit thread level parallelism in applications. They partition/share the resources present on chip for one physical core into two logical processors. This way the users and the Operating system see 8 cores in a 4-physical core machine. SMT is used in today's processors irrespective of whether it is CMP or SMP.

Figure 1 shows an example of different configurations of a Multiprocessor. The figure depicts two NUMA nodes

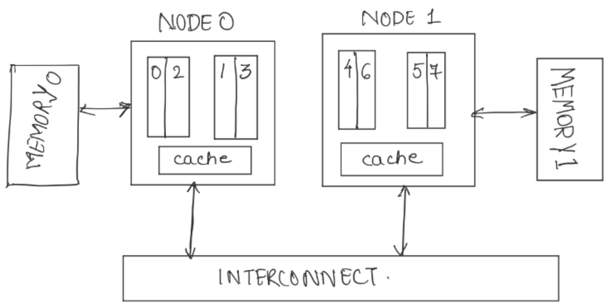

Figure 1: Different type of SMT, CMP and SMP configurations

or here in this 2 Sockets on a SMP distributed memory multiprocessor. The processors 0-3 belong to Socket 0/Node 0 and 4-7 belong to Socket 1/Node 1. Each NUMA node has its own memory bank. So the access time of CPU 0-3 to Memory 0 is less than the access time of CPU 4-7 to Memory 0. However, the access time of Node 0 to Memory 0 is same as access time of Node 1 to Memory 1, hence the name Symmetric.
The Node 0 depicted in the figure is a CMP with 2 physical cores and 4 logical cores. The cores (0, 2) and (1, 3) are multi-threaded. The figure is just an example of a simple system. The Sockets individually can have multiple NUMA nodes as well with different memory banks. In our example, sockets and NUMA nodes are same.

Two processes running on CPU 0 and CPU 1 would only share the memory resources and not the CPU resources. However, two processes running on CPU 0 and CPU 2 would share both the CPU and memory resources so their execution time would increase because of resource contention.

Two processes running on CPU 0 and CPU 4 that is CPU's on different sockets would be sharing only the off-chip interconnect and in some case share memory locations across banks. They have different memory banks and different CPU resources. Hence the processes run faster than CPU 0 and CPU 1. We will see this in Section 3.

For running our benchmarks on different configurations we pin benchmarks to different cores according to the sharing we want. Our Target machines are:

- **CMP** : Intel's (i7-7700 **3.60GHz**) 4 physical cores with 2 threads per core. There is only one NUMA node so all processors have uniform access time to the global memory. There are 8 cores 0, 1, 2, 3, 4, 5, 6, 7. The logical core pairs (SMT cores) are (0, 4), (1, 5), (2, 6), (3, 7).

  We run the following configurations on this machine:

  - **<u>Reference baseline</u>**: We run a single copy of the benchmark on a single core of the target machine with all resources (no sharing).
  - <u>CMP</u> : We pin two copies of benchmarks on the two physical cores 0 and 1. So both the processors will share only the memory/cache bandwidth and not the CPU resources.
  - <u>SMT</u>: We pin two copies of benchmarks on the two logical threads of the same physical core i.e CPU 0 and CPU 4. So both the benchmarks will share the same CPU resources as well as the memory bandwidth.

- **SMP0 (Single Socket with NUMA nodes)**: This is a variant of SMP with one socket but multiple NUMA nodes.

  AMD (EPYC 7551P **2.5GHz**), 32 physical cores with 2 threads per core, 4 NUMA nodes connected by on-chip interconnect (Name of server - Glados).

  - 0-31 are physical cores (one CMP)
  - 32-63 are their logical counter parts. So, CPU 0 and CPU 32 are two logical cores on the same physical core and so on.
  - NUMA node0 CPU(s): 0,4,8,12,16,20,24,28, 32,36,40,44,48,52,56,60
  - NUMA node1 CPU(s): 1,5,9,13,17,21,25,29, 33,37,41,45,49,53,57,61
  - NUMA node2 CPU(s): 2,6,10,14,18,22,26,30, 34,38,42,46,50,54,58,62
  - NUMA node3 CPU(s): 3,7,11,15,19,23,27,31, 35,39,43,47,51,55,59,63

  We run the following experiments on this machine:

  - We pin two copies of the benchmarks to **two CPU's on the same NUMA nodes** like CPU 0 and CPU 4. In this configuration, they will be sharing the same memory bank.
  - We pin two copies of the benchmarks on **two CPU's in different NUMA nodes** like CPU 0 and CPU 1. So they will both access different memory banks.

- **SMP1 (Multiple Socket)**: This is SMP machine with 2 sockets. Intel (Xeon Silver 4116 **2.10GHz**), 24-cores with 2 threads/core machine, 2 sockets (chips) with 12 physical cores each. (Name of Server - BigBox1)

  - 0-23 are physical cores. 0-11 on Socket 0 and 12-23 on Socket 1.
  - 24-47 are their logical counterparts. So CPU0 and CPU 24 are the logical cores on the same physical cores.
  - NUMA node0 CPU(s): 0-11,24-35
  - NUMA node1 CPU(s): 12-23,36-47

  We run the following experiments on this machine:

- We pin two copies of the benchmarks to **two CPU's on the same socket** like CPU 0 and CPU 1. In this configuration, they will be sharing the same memory bank.
- We pin two copies of the benchmarks on **two CPU's in different sockets** like CPU 0 and CPU 12. So they will both access different memory banks.

2.1 About SPEC CPU 2017

We are using SPEC CPU 2017[1] benchmarks to study their performance on different multiprocessor architectures. SPEC benchmarks are industry standard, CPU-intensive suites used to study the compute intensive performance by stressing the system's CPU, memory subsystem and compiler. The SPEC 2017 benchmarks contain around 47 benchmarks categorized among four different suites:

- SPECspeed Integer
- SPECspeed Floating Point
- SPECrate Integer
- SPECrate Floating Point

SPECspeed suites are used to study the time taken by the computer to perform single tasks. They always run only one copy of each benchmark. Higher scores of rate indicate that less time is needed to run the task.

SPECrate suites are used to study the throughput obtained that is the work done per unit time. They run multiple concurrent copies of each benchmark. Higher scores of rate indicate higher throughput obtained.

There are different benchmarks for different applications like data compression, image compression, 3D rendering, image manipulation application, etc. These applications have different compute and memory requirements which directly affects their execution time on a real machine. We have based our study on 7 different benchmarks with different characteristics and reported their results in Section 3.

We have run all our benchmarks in the Base mode with common compiler optimizations. We have not tested Peak mode that can use different compiler optimizations for different benchmarks.

2.2 Benchmarks Used

We have used the following benchmarks[1,2] for our study:

- **538.imagick_r**: It is an image manipulation *floating-point* application. This benchmark has *high instruction dependencies* which are the major cause of pipeline stalls and takes a large part of the execution time. Memory access pattern is mostly *L1 bound.*
- **557.xz_r**: It is a general data data compression *integer* application. This benchmark suffers from a *high branch mis-prediction rate* hence their execution time is affected by front-end stalls. The memory access pattern is mostly *L3 and DRAM bound.*
- **503.bwaves_r**: This benchmark numerically simulates blast waves in three dimensional transonic transient laminar viscous flow.It is a *floating-point* benchmark. Most of the execution time for this is spent in instruction retirement and *high instruction dependencies.*
- **49.fotonick3d_r**: This benchmark is used in Computational Electromagnetics. It is a *floating-point*benchmark. It is mostly *DRAM Bound.*
- **541.leela_r**: This is an *integer* benchmark used in Artificial Intelligence for Monte Carlo simulation, game tree search & pattern recognition. It suffers from *high branch mis-predictions* hence most of the execution time is spent on front-end stalls. The memory access is mostly *L1 bound.*
- **548.exchange2_r**: This is a Sudoku Puzzle generator. Its a *floating-point* benchmark. This Application is mostly *Retirement bound.*
- **520.omnetpp_r**: This *integer* benchmark performs discrete event simulation of a large 10 gigabit Ethernet network. This benchmark is mostly *DRAM bound.*

3 Results

In this Section, we will see how the properties of benchmarks described in Section 2.2 have an effect on the behavior of the benchmark on different machines and how much slowdown each configuration shows.

Figure 2 and Figure 3 shows the execution time slowdown rate and slowdown in the BW rate respectively, that is achieved on SMT, CMP and SMP0 and SMP1 machines with respect to our baseline. We report our results for Baseline, SMT cores, CMP cores, SMP0 cores on different NUMA nodes(**SMP0_DN**), SMP0 cores on the same NUMA node(**SMP0_SN**), SMP1 cores on different sockets(**SMP1_DS**) and SMP1 cores on the same socket(**SMP1_SS**).

The slowdown achieved in the throughput is just the reverse behaviour of the slowdown achieved in the Execution Time as is shown in Figure 3.

3.1 Slowdown on SMT and CMP

- **Slowdown on SMT:** The figure shows that there is always a slowdown between SMT cores and the baseline. This is because SMT cores share the physical CPU resources among the two processes running on them. We see the worst slowdown of approximately 2.1x on SMT cores for the benchmark

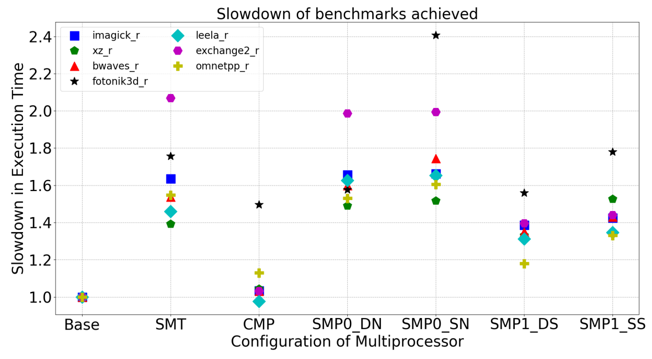

Figure 2: Slowdown in Execution Time SMT, CMP and SMP configurations

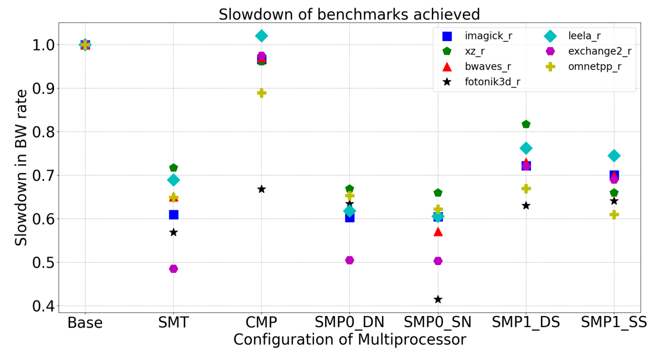

Figure 3: Slowdown in BW achieved on SMT, CMP and SMP configurations

Exchange2_r. This is because this benchmark is bounded by computation resources as seen in Section 2.2 so CPU resource sharing has the worst impact on it.
The other benchmarks show a slowdown between 1.5x to 1.7x on SMT.

- **Slowdown on CMP:** The benchmarks running on 2 CMP cores have their own CPU resources but share the memory bandwidth.The slowdown on CMP is not very significant for benchmarks since there is no CPU resource sharing. However, the benchmarks like **fotonik3d_r** and **omnetpp_r** which are DRAM bound show a slowdown of 1.5x and 1.2 x respectively on CMP cores because the processes share the memory bandwidth.
Some benchmarks which are front-end bound actually give better performance on CMP cores.

- **Slowdown comparison between SMT and CMP:** The maximum comparative slowdown of around 2.1x between CMP and SMT is shown by **exchange2_r** because resource sharing bounds the execution time. Hence we see a direct impact of resource sharing on performance of compute intensive benchmarks.

  Fotonik3d_r shows the least variation in the slowdown between SMT and CMP since it is DRAM bound and the bottleneck is memory and not computations.

  The other benchmarks show similar comparative slowdown between SMT and CMP of around 0.5x-1x.

3.2 Slowdown on SMP

The slowdown on the SMP (SMP0 and SMP1) machines are because of the organization of the memory and the cores/CMP's connected via on-chip or off-chip interconnects. Some inherent slowdown is also because the machines used for SMP are of lower CPU frequency than the CMP/SMT machines. So we have not made apple-apple comparison between CMP and SMP results in our report**.
***** SIDE NOTE:** We also tried to take another baseline reference on the SMP machine and calculate slowdown for SMT and CMP. The rate of slowdown obtained between the new baseline and SMT (run on SMP machine) is the same as the rate of slowdown obtained on the CMP machine.* **This shows that the effect to the performance of benchmarks to the extent of resource-sharing is similar no matter where the baseline was run.**

As shown in Figure 2, we have compared the performance of benchmarks run on different NUMA nodes and different sockets CPU's with the performance of the same benchmarks when run on same NUMA node or same Socket CPU's.

Slowdown comparison between same (SMP0_SN) and different NUMA nodes (SMP0_DN) done on SMP0:

- For **exchange2_r**, we don't see any difference in slowdown between SMP0 cores on different and same NUMA nodes because this benchmark is not memory bound.

- For **fotonik3d_r**, we see the maximum difference in slowdown between SMP0 cores on different NUMA nodes (1.6x) and SMP cores on the same NUMA nodes (2.4x). This is because the benchmark is DRAM bound so it shows more slowdown on the SMP cores on the same NUMA node where the processors share the memory bandwidth.

- We also conclude that not all the benchmarks that are DRAM bound perform in the same way. For example, both *omnetpp_r* and *fotonik3d_r* are DRAM bound. However, the difference in slowdown is more significant for *fotonik3d_r*.

Slowdown comparison between same (SMP1_SS) and different socket cores (SMP1_DS) done on SMP1: The resource that is shared between the CPU's on the same socket is mostly the memory bandwidth. This is evident in our results.

- The benchmarks like **fotonik3d_r**, **omnetpp_r** and **xz_r** that are DRAM and L3 bound, show the maximum variation (of around 0.2x) in slowdown rates between the same socket CPU's and different socket CPU's. They perform worse when they are contending for the same memory bank resources i.e. on the same socket.

- The other benchmarks are mostly performing equivalent on both the configurations.

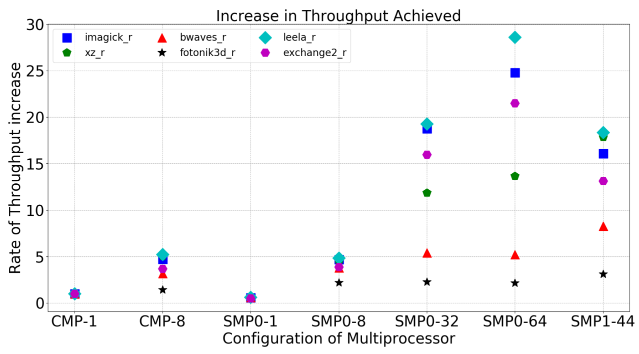

Figure 4: Increase in Throughput achieved running multiple copies of benchmarks on CMP and SMP

Figure 4 shows the increase in throughput achieved when we run multiple copies of the benchmarks on the machines. We run 8, 32, 44 and 64 copies of the benchmarks on the CMP and SMP0 and SMP1 machines to study the throughput achieved by stressing the system completely.

Observations:

- The maximum throughput achieved when we run 8 copies of the same benchmark on a 8-core CMP machine is 5x with respect to the baseline (1 copy on 8-cores).

- We achieve a 5x increase in the throughput when we run 8 copies of the benchmark on a 64-core SMP0 machine. This is because the OS schedules a single copy of the benchmark on one core so it only uses 8 cores out of 64 cores. Hence, we see an equivalent throughput as a CMP machine with 8-cores only. The performance would have been better if these benchmark programs were multi-threaded but the SPECrate benchmarks don't support OpenMP threading. Only a few SPECspeed benchmarks support threading. But that is out of the scope of this report.

- However, the throughput increase on the 64-core SMP0 machine varies a lot between benchmarks when we run 32 and 64 copies on it. The throughput increase is maximum, upto 28x for the benchmarks that are not DRAM bound (eg. *leela_r* and *imagick_r*). This is logical because they face less memory contention on a stressed machine hence achieve higher throughput. The DRAM bound benchmarks face more contention on a stressed SMP machine.

- We also observe that not all all benchmarks with the same characteristics achieve the same increase in throughput. For example, both *leela_r* and *xz_r* are integer benchmarks bounded by front-end stalls due to branch mis-predictions. However, *leela_r* shows upto 15x higher throughput than *xz_r*. This is because *xz_r* also has a significant part of its execution time bounded by accessing L3 and DRAM whereas *leela_r* is only L1 bound which is usually local to the CPU core on which the benchmark is running. Hence, it is able to achieve a higher bandwidth even on a completely stressed machine.

4 Conclusions

This report gives an analysis of how the benchmark performance change on different multiprocessor machines. We also show that the slowdown because of the sharing of CPU resources, memory and interconnect is dependent on the workload characteristics. When we run a compute intensive workload on cores which share compute resources, the slowdown would be significant. When we run a memory-intensive workload on cores that share memory bandwidth, then the slowdown is significant.

5 References

1. SPEC CPU 2017 benchmarks

2. Experiments with SPEC CPU 2017