A Shared Memory Benchmark in OpenMP

Matthias S. Müller[1]

High Performance Computing Center Stuttgart (HLRS), Allmandring 30, 70550 Stuttgart, Germany,
mueller@hlrs.de,
WWW home page: http://www.hlrs.de/people/mueller

Abstract. The efficient use of the memory system is a key issue for the performance of many applications. A benchmark written with OpenMP is presented that measures several aspects of a shared memory system like bandwidth, memory latency and inter-thread latency. Special focus is on revealing and identifying bottlenecks and possible hierarchies in the main memory system.

1 Introduction

With the introduction of OpenMP the portability and ease of shared memory programming has been greatly improved. The concept is simple. All data can be accessed by every thread and the programmer can focus on the distribution of workload. Despite this simplicity many issues will influence the performance and several benchmarks have been designed to address them [4, 6, 1, 3, 5]. Since many scientific applications are memory bound the performance will strongly depend on an efficient use of the shared memory. Many issues will have an influence on the performance. Is it a flat SMP or hierarchical NUMA system? What are the different bandwidths and how is the memory distributed? Are there any bottlenecks I have to be aware of. Instead of looking at the memory system with complete applications [7] this benchmark tries to answer these or similar questions. Since HLRS has many scientific and industrial users with codes that are not cache friendly, the focus is on the main memory system. All tests are designed to run out of cache.

Since OpenMP provides a rather abstract interface to parallel programming that is not very hardware oriented, writing a shared memory benchmark is not trivial. Not only that it is undefined where the allocated memory is located physically and relative to the threads. The operating system is also free to schedule the threads to the available CPUs with its own policy. Both assignments may change during runtime. It is therefore unclear whether a portable benchmark written in OpenMP may reveal any information besides average values for a given architecture. The pragmatic approach taken here is, that if the operating system is smart enough to hide something from a good benchmark it should also not be relevant for a real application. As examples for different kind of shared memory systems the results of three different platforms are presented:

PC A dual processor Pentium IV pc with 2 Gigabyte of RDRAM memory. It serves as an example for a simple bus based system.

Hitachi SR8K The Hitachi SR8000 consists of nodes. Each node is a flat SMP system with eight CPUs.

HP N-Class This system also has eight CPUs, but the system is organized in cells. Each cell has local memory, the connection to remote memory is made via a fast interconnect.

2 Latencies

Since different threads communicate with each other by reading and writing shared memory, the latencies involved with these 'communications' are an important factor for the overall performance. Two different kind of latencies can be distinguished. First the latency to access the main memory, second the latency that occurs in the direct 'communication' between two threads.

2.1 Latency to Main Memory

With the rapidly increasing clock frequency the latency to the main memory measured in clock cycles is getting higher and higher. Therefore techniques have been developed to reduce or hide this latencies. Examples are caches, out of order execution or data prefetching. Nevertheless, as soon as these optimization techniques fail the cost of a memory access is dominated by the latency.

To measure the memory latency the core loop looks like this:

```
for(i=0; i<N; i++) {
   idx = index[idx];
}
```

The index field is initialized such that every access reads a different cacheline. The total memory amount that is referenced exceeds the cache size. The structure of the loop and the initialization of the index field should inhibit any optimization techniques. In Fig. 1 different latencies are shown. For the results labeled `serial` everything was measured in a serial region, `offset` denotes the owner of the indexfield that is used. A smart operating system could move the running thread to a CPU that is close to the memory or migrate pages. This is one of the reasons why for all other results all threads are measuring the latencies simultaneously. A comparison with the serial results should reveal several informations. If all observed latencies are significantly higher than in the serial case it indicates a limitation in the transaction system. If suddenly a two-level hierarchy is visible the optimization that acted in the serial case now fails.

The results of the HP N-Class are shown in Fig. 1. There is no visible hierarchy. When multiple threads are active the latency increases from about 250ns to 300ns. The same behavior is shown by the PC, where the latency increases from 170ns to 210ns. The latency of the Hitachi SR8K is around 390ns independent of the number of active threads.

2.2 Interprocess Memory Latency

In a Shared Memory Programming Model threads communicate with each other by sharing variables. The question posed here is how long it takes before a modification made by one thread is visible to another thread. If the information exchange must be made by reading and writing from main memory the answer would be simple: two times the latency to main memory for the half trip time. One thread would write the memory, another would read it. In real life the situation is more complex. Since the two threads are not synchronized half a memory latency is added on average. The write latency could be higher than the read latency, because on some systems the whole cache line has to be read before it can be written. Altogether this results in a round trip latency between 4 and 7 memory read latencies as measured in the previous section. However, in a bus based system the second thread could fetch the cache line while it is written. Two CPUs might share the same cache or the two threads may run on the same CPU.

To measure the latency between two threads a simple ping-pong style benchmark is used.

```
rank=omp_get_thread_num();
if(rank == sender ){
  for(i=0;i<count ; i++){
    /*send:*/
    token=1;
    #pragma omp flush(token)
    /*receive:*/
    while(token != 0){
       #pragma omp flush(token)
    }
  }
}
if(rank == receiver){
  for(i=0; i<count ; i++){
    /*receive:*/
    while(token != 1){
       #pragma omp flush(token)
    }
    /*send:*/
    token=0;
       #pragma omp flush(token)
  }
}
```

The program is running with N threads. On all systems the number of threads was chosen equal to the number of CPUs. In order to observe possible hierarchies in the system the ping pong benchmarks is made between all

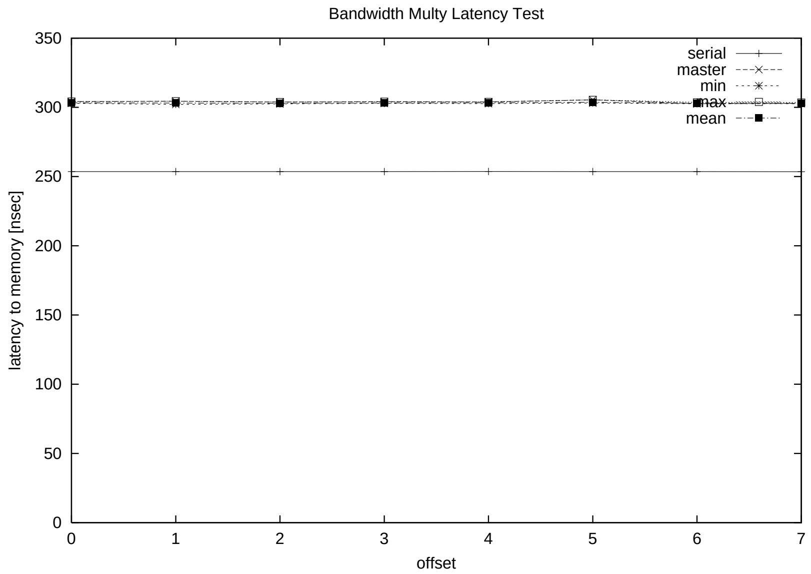

Fig. 1. Latency to Main Memory for HP N-Class. There is no visible hierarchy. When multiple threads are active the latency is 300μs instead of 250μs.

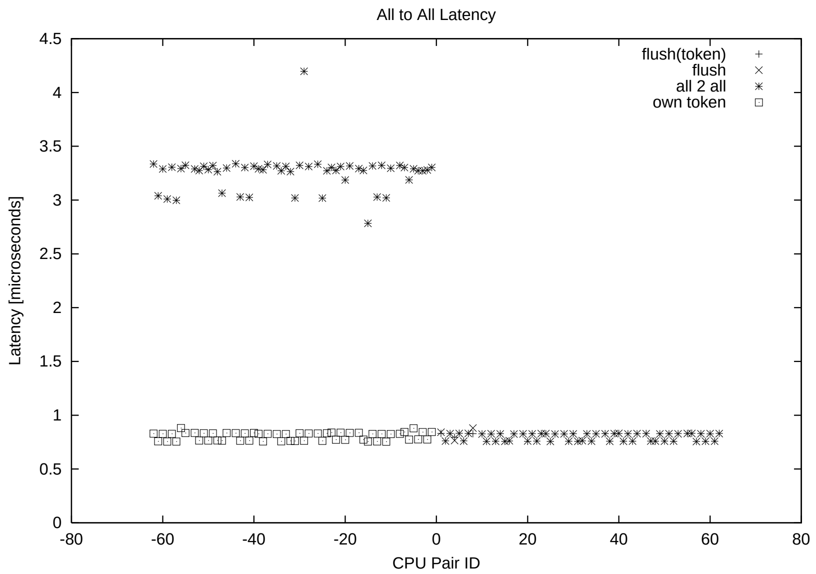

Fig. 2. Ping Pong Latencies for HP N-CLASS.

$N*(N-1)$ pairs of threads. To identify a pair consisting of two threads with the thread number $n1$ and $n2$ the value $N*n1+n2$ is assigned to the pair.

The code fragment above shows the original form of the benchmark. Several variants are used to examine more details. First the call to `omp flush(token)` is replaced by a call to `omp flush`. After this call a consistent view of the complete memory is guaranteed which makes it potentially more expensive. In the second modification there is not one single token, but the token used in the communication between two threads is always owned by one of the threads. Finally, in the all to all case $N/2$ pairs are communicating simultaneously. Each pair has its own token, which is owned by one of the threads.

In Fig. 2 the latencies of the HP N-Class are plotted vs. the pair ID. Negative thread IDs are used for the last two modifications. For all modifications a two level hierarchy is visible, for the all to all benchmark the latencies are significantly increased. The observed latencies are 3000ns/3300ns (10 read latencies) and 760ns/830ns (3 read latencies)for the other tests. The SR8K shows a inter-thread latency of 2000ns (5 read latencies) and the PC of 280ns. The PC is the only platform where the inter-thread latency is significantly lower than the predicted value of four to seven times the read latency. This indicates an active bus snooping. The all to all communication reveals again a bottleneck of the HP N-Class.

3 Bandwidth

To measure the memory bandwidth the access patterns from the well known stream benchmarks [2] are used. Reading from and writing to memory is measured with `SUM` and `FILL` respectively. `COPY` reads and writes at the same time, while `DAXPY` performs the operation $a(i) = a(i) + qb(i)$, with two loads and one store. The length of the vectors a and b are choosen such that all operations run out of cache. The unmodified stream 2 benchmark that measures memory performance for different vector lengths is used to select an appropriate vector length.

To achieve a high bandwidth a good memory subsystem and an appropriate connection of the CPU is required. In the shared memory case the requirements are more stringent. The memory system has to fullfil the concurrent requests of several CPUs. Any bottleneck will result in a loss of performance at some place. The purpose of the next two benchmarks is to find such bottlenecks.

3.1 Scale-up with Number of Threads

To avoid any overhead that might be introduced by OpenMP or inter-thread synchronization, each threads works on its own piece of memory. The memory is allocated and initialized in the parallel section by each thread. This should give optimal results if the memory is owned by the allocator or if a first touch algorithm is in place.

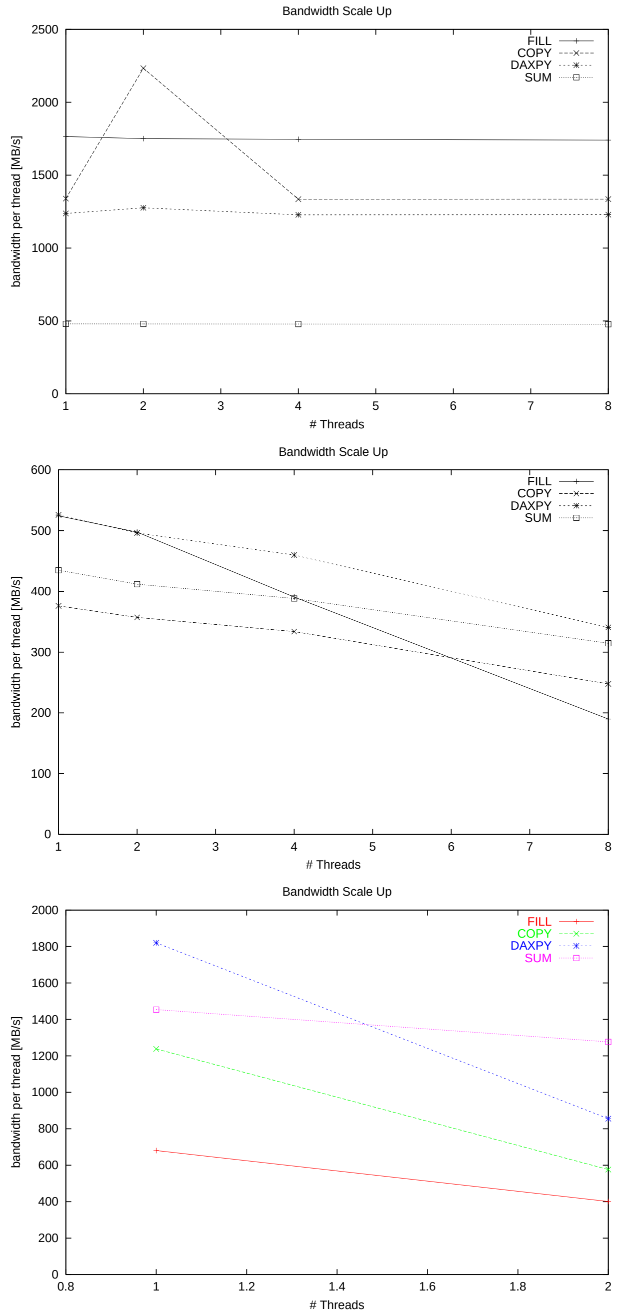

Fig. 3. Scale-up of Memory Bandwidth for Hitachi SR8K, HP N-Class and the PC (from top to bottom).

Each thread works independently, there is no OpenMP work-share directive. This should keep the likelihood of compiler performance bugs to a minimum. Only barriers between the different tests are introduced to synchronize the tests. The results are visible in Fig. 3. For a simple bus based system like the PC, the bus is already saturated by one CPU. For a cell based system like the HP N-Class the results are better, but the limitations of the memory bandwidth are clearly visible. The memory system of the SR8K scales perfectly, with a constant memory bandwidth per thread.

3.2 Speed-up with Number of Threads

The disadvantage of the previous benchmark is that it could also be performed by a distributed memory system or at least a virtual shared memory system with good results, since the threads act on completely different memory. Therefore we test a modification where the memory amount is kept constant and all threads work on the same large piece of memory in an OpenMP parallelized loop.

Since the amount of memory and the resulting work load is high enough, the overhead of the OpenMP parallelization is negligible. For the systems tested here, there should be no significant difference in the capability to scale with the number of threads compared to the previous test. The results of the systems presented here reflected the expected behavior. Differences in the results of speed-up and scale-up tests should be carefully analyzed not to be performance bugs in the OpenMP compiler.

3.3 Scaling with Memory Amount

Another test that may reveal structures in the memory system is the scaling of the bandwidth with the used memory amount. Since the start value is already much larger than the cache size, the bandwidth should remain constant for a flat SMP system. For a cell based system the cell structure could become visible as more and more of the systems memory is used. The total available bandwidth increases as soon as more than one cell is used, depending on the allocation strategy and the inter cell memory bandwidth. Finally, if the system is low on memory it puts additional pressure on the memory system. It may for example prevent page duplication. The PC, HP N-Class and the SR8K did show a constant bandwidth.

3.4 Memory Granularity

In this benchmark the memory amount and the number of threads is kept constant. To put an additional pressure on the memory system, the size of the requested memory pieces are reduced. This is achieved by choosing a `STATIC` schedule with a `CHUNKSIZE` that is modified from one to a value where each thread gets one chunk.

The results are visible in Fig. 4. As expected the performance is best for a large chunksize. If the chunksize is reduced, the bandwidth first remains high.

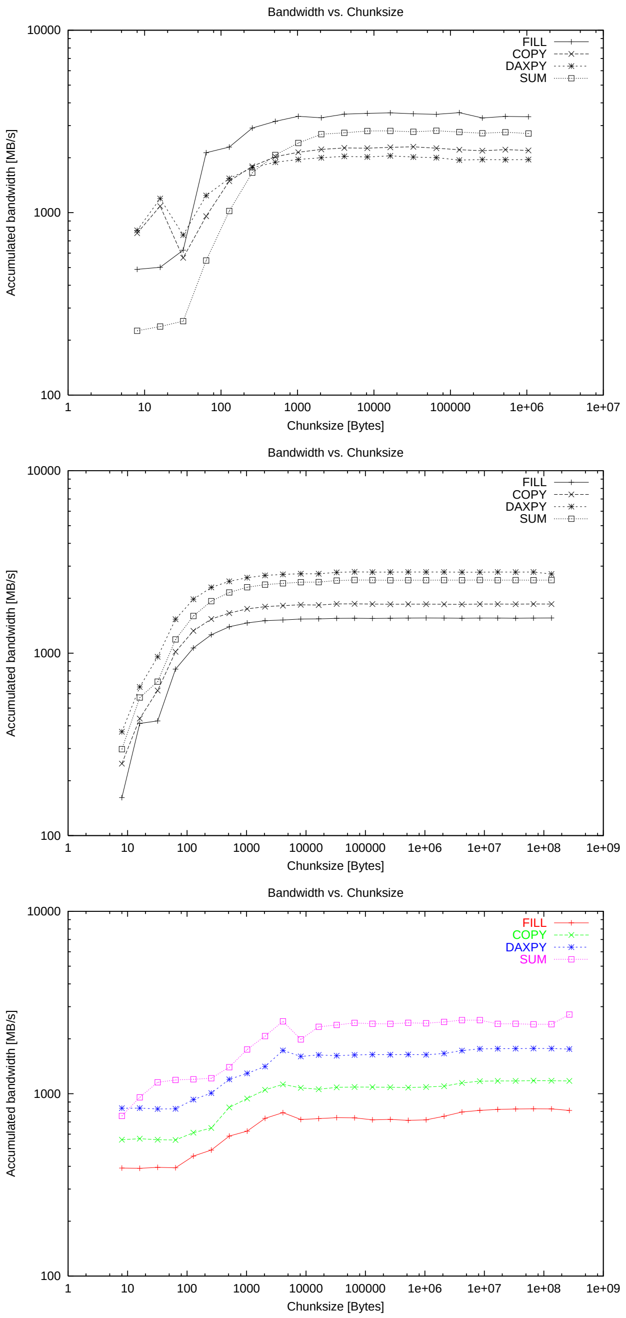

Fig. 4. Memory Bandwidth vs. Chunksize for Hitachi SR8K, HP N-Class and the PC.

This indicates that the parallel overhead remains constant due to the static schedule that was chosen. If the chunksize is smaller than a critical value the performance gets lower. For all systems the size of this value is consistent with the page size.

The HP N-Class shows a steep descent of the performance down to the smallest chunksize of one double value. The other two systems show an almost constant performance for very small chunksizes. Since the OpenMP standard does not provide an exact definition of the schedule policies, this behavior can not be deduced to an architectural feature. The standard states that 'a program should not rely on a schedule kind conforming precisely to the description given'. This gives the compiler the freedom to choose a minimum chunksize that corresponds to a cacheline. An own SPMD style programming without using work-share directives would prevent this, but the resulting more complicated loop structure may inhibit optimization techniques.

4 Summary of Results

For all benchmarks the temptation is high to come up with one single number that could be used to rank systems. However, the information that is lost by reducing the results to a single number is often high. Nevertheless, as an overview and to serve as a starting point for a deeper look into the results a summary would be beneficial.

Since it is difficult to compare systems with different numbers of CPUs the results for different thread numbers are provided. It also allows to check whether the system really scales to the maximum number of threads. Each result is the mean value of the bandwidth per thread that was achieved in the single tests.

To come up with an overall effective bandwidth all latencies have to be converted to a bandwidth. This is done by artificially assigning a memory amount of 8 bytes (one double value) to each transaction. A latency Δt converts to a bandwidth of $8Bytes/\Delta t$. As a final result for each number of threads the geometric mean of all test results is given.

An example is the following result achieved by the HP N-Class:

threads:	1	2	4	8
speed up :	355.953595	343.561199	325.763491	267.504597
scale up :	465.286751	440.674049	393.185014	273.207580
chunksize :	373.017109	341.720323	312.215239	233.193936
memory scale :	351.912566	338.211988	317.428108	264.067959
latency :	30.385659	28.412623	28.260233	25.982235
ping pong :	0.386203	0.386203	10.813685	1.953931
geometric mean :	79.639219	75.954131	125.350109	78.188061

The bottlenecks in the systems bandwidth are clearly visible. The overall value is dominated by the low results of the latency and ping pong tests due to the nature of the geometric average. A simple mean value, on the other hand, would be dominated by the larger number of bandwidth tests.

5 Conclusion and Outlook

The presented tests reveal a number of properties and bottlenecks of the system under investigation. The memory and inter-thread latency benchmarks did clearly reveal hierarchical systems. They also provide information about the costs of a memory access in a worst case scenario.

The scale-up test shows the scalability without interference with OpenMP overhead or compiler issues. In this sense the speed-up test is targeted to the same bottlenecks with a higher requirement for the compiler and run-time library. The variation of chunksizes shows the reaction of the memory system if the requested memory chunks are reduced. Although it already provides some insight, its exact interpretation is limited by the tolerant definition of the scheduling clause by the OpenMP standard. The scaling of the memory amount is targeted to systems where the operating system is not aware of a cell structure of the underlying hardware.

The benchmarking of further systems in the near future will show if these set of tests is sufficient to provide all relevant information about the shared memory system. Further tests are already in development, but in a too early stage to be presented here. A larger number of tests will also show whether the proposed benchmark summary is useful for a comparison of different systems.

Acknowledgements

I wish to thank Uwe Küster (HLRS) and Holger Berger (NEC) who provided the idea and basic code for the measurement of the memory latency.

References

1. J. M. Bull. Measuring synchronization and scheduling overheads in OpenMP. In *First European Workshop on OpenMP*, 1999.
2. John D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE TCCA*, Dec. 1995. http://www.cs.virginia.edu/stream.
3. Matthias Müller. Some simple OpenMP optimization techniques. In R. Eigenmann and M.J. Voss, editors, *OpenMP Shared Memory Parallel Programming*, volume 2104 of *LNCS*, WOMPAT 2001, West Lafayette, IN, USA, 2001. Springer.
4. RWCP. Openmp version of NAS parallel benchmarks. http://pdplab.trc.rwcp.or.jp/Omni/benchmarks/NPB/index.html.
5. Mitsuhisa Sato, Kazuhiro Kusano, and Sigehisa Satoh. Openmp benchmark using PARKBENCH. In *Proceeding of Second European Workshop on OpenMP*, Edinburgh, Scotland, U.K., Sept. 2000.
6. SPEC. SPEComp 2001. http://www.spec.org.
7. Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The splash-2 programs: Characterization and methological considerations. In *Proceedings of the 22nd int. Symp. on Computer Architecture*, pages 24–36, June 1995.