In *Proc. Workshop on Applications for Multi and Many Core Processors (A4MMC) at ISCA 2011*, June 2011

Impact of System and Cache Bandwidth on Stencil Computations Across Multiple Processor Generations

Robert Strzodka
Max Planck Institut Informatik, Germany

Mohammed Shaheen
Max Planck Institut Informatik, Germany

Dawid Pająk
West Pomeranian University of Technology, Poland

Abstract

We compare old single-core multi-processor systems against multi-core processors and study the question which improvements are most relevant for increasing the performance on stencil computations. Even before the multi-core era began, the bandwidth wall*, the discrepancy between off-chip bandwidth requirements and system bandwidth performance, was already a significant problem. Because of the currently growing number of parallel cores in CPUs this discrepancy could only be stopped from further deterioration by introducing dual-, triple- and quad-channel memory interfaces. However, this type of off-chip bandwidth scaling is too expensive and thus only a temporary relieve that cannot keep up indefinitely with the exponentially growing number of cores. Therefore, we analyze in particular how the scaling of system and cache bandwidth affects the performance of stencil computations.*

We evaluate both naive stencil implementations as well as time skewing variants that exploit temporal locality and minimize the number of cache misses in case of iterative stencil computations. We prove certain invariance properties of the schemes and develop a corresponding performance model. Then, we use this model to find out which hardware improvements in the old single-core processors are necessary to match the performance of the new multi-core processors. From this we can draw conclusions about most effective improvements for future processors.

1 Introduction

In stencil computations, a small neighborhood of each element in a domain contributes to the new value of this element. In the simplest case we have constant weights which are multiplied with the neighbor elements and summed up, e.g. in 3D a general 7-point stencil has a weight for the element itself and 6 weights for each of its direct neighbors in the 3 spatial dimensions x, y and z:

$$\begin{aligned} v'(x,y,z) \quad = \quad & w_1\, v(x,y,z) + w_2\, v(x-1,y,z) + w_3\, v(x+1,y,z) + w_4\, v(x,y-1,z) \\ & + w_5\, v(x,y+1,z) + w_6\, v(x,y,z-1) + w_7\, v(x,y,z+1)\,. \end{aligned}$$

This computational pattern occurs very often in scientific computing because discretized differential operators take this form (with possibly spatially varying weights $w_k(x,y,z)$). A big problem with these operations is the low arithmetic intensity, performing just one multiply-and-add operation for each retrieval of one neighbor element.

The cache helps in this scenario to retrieve the x-neighbors $v(x-1,y,z), v(x+1,y,z)$ (unit stride) quickly. Keeping entire lines of the domain in the cache, the y-neighbors

Alg. 1 Cache accurate time skewing in 3D. Only few transformations are necessary to obtain much faster parallel C++ code from the naive implementation. Function stencil_SSE() contains the stencil computation vectorized along the x-axis from 0 to WIDTH. The tile sizes are chosen such that they fit into the last level cache, more details are given in [9].

```
void NaiveSSE_3D ()
{
  for(int t = 0; t < T; t++) {
    for(int z = 0; z < DEPTH; z++) {
      for(int y = 0; y < HEIGHT; y++) {
        stencil_SSE(t, z, y, 0, WIDTH);
      }//y
    }//z
  }//t
}

void CATS_3D (int threadID)
{
  for( TileIt tile = tileSet[threadID].begin();
       tile != tileSet[threadID].end(); ++tile) {
  wait_on_dependencies(tile);

  for(int z = 0; z < DEPTH; z++) {
    for(int t = tstart(tile,z); t < tend(tile,z); t++) {
    for(int y = ystart(tile,z,t); y < yend(tile,z,t); y++) {
      stencil_SSE(t, z-s*t, y, 0, WIDTH);
    }}//t,y
  }//z

  }//tile
}
```

$v(x, y-1, z), v(x, y+1, z)$ can also be fetched from cache after reading them once, and for large caches that hold three 2D slices of the domain even the z-neighbors $v(x, y, z-1), v(x, y, z+1)$ have to be read only once. From there on, however, the execution time does not improve if we make the cache even bigger, because no additional data reuse can occur. Only if the cache size becomes large enough to hold the entire domain, then we could benefit in multiple applications of the stencil, which would all happen in cache.

While more complex work-loads often obey a power law for cache misses [3] with a continuous benefit from cache size increases, for the stencil computation we only gain at discrete thresholds when the unit stride neighbors, the neighboring lines, the neighboring 2D slices or the entire domain fits into the cache. With typical cache and domain sizes, we find ourselves in the situation that the neighboring 2D slices fit, but the domain is much larger than the cache. So all neighbors come from the cache, but every element $v(x, y, z)$ has to be read once from system memory for every application of the stencil. So the achieved execution time for the entire 3D domain remains restricted by the system bandwidth.

This fact is particularly painful if the stencil application is repeated hundreds of times, as happens in iterative solvers of linear equation systems, or the discrete time evolution in partial differential equations. In each iteration the entire domain has to be fetched from system memory in the naive implementation. We cannot solve the problem with more advanced cache hardware. Even latest plans for 3D-stacked cache [4, 11] would not help here if they do not hold the entire domain. Hardware tradeoffs offered by the continuous interpretation of the power law [6] do not apply. Similarly, clever cache utilization strategies [1, 7] cannot improve the execution time as we already know in advance which elements we need for each processing step. But these proposed approaches become useful

Table 1 Hardware configurations of our workstations. The first half of the table refers to the technical specification of the CPUs. The second half presents results of synthetic benchmarks on these machines.

Brand	Intel	AMD	Intel	AMD	Intel
Processor	Xeon MP	Opteron 250	Core i5 Sim	Opteron 2218	Core i7 940
Code-named	Gallatin	Troy	-	Santa Rosa	Bloomfield
Frequency	3.06 GHz	2.4 GHz	2.93 GHz	2.6 GHz	2.93 Hz
Number of sockets	2	2	1	2	1
Cores per socket	1	1	2	2	4
L1 Cache per core	8 KiB	64 KiB	32 KiB	64 KiB	32 KiB
L2 Cache per core	512 KiB	1024 KiB	256 KiB	1024 KiB	256 KiB
L3 Cache per core	1024 KiB	-	4096 KiB	-	2048 KiB
Number of threads	2	2	2	4	4
Measured L1 Bandwidth	44.4 GB/s	36.4 GB/s	91.1 GB/s	79.3 GB/s	182.3 GB/s
Measured L2 Bandwidth	24.3 GB/s	21.0 GB/s	61.3 GB/s	40.6 GB/s	124.0 GB/s
Measured L3 Bandwidth	17.3 GB/s	-	44.5 GB/s	-	88.2 GB/s
Measured Sys. Bandwidth	3.2 GB/s	6.4 GB/s	14.4 GB/s	11.2 GB/s	17.8 GB/s
Last Level Cache/Sys. Band.	5.4	3.3	3.3	3.6	3.6
Measured Peak DP FLOPS	11.5 G	9.6 G	19.6 G	20.1 G	39.1 G
Measured Stencil DP FLOPS	7.9 G	4.4 G	12.2 G	11.1 G	24.5 G

if we first reformulate the problem on the algorithmic level.

The key idea in efficient iterative stencil computations is to perform multiple iterations locally on small tiles of the domain. Then data in the cache can be reused for multiple iterations without going to the system memory. Because of the dependence on neighbors in each iteration step, multiple local iterations enforce index dependencies in such computations. They are known as *time skewing* techniques [8, 13, 14]. Because more data is read from the cache, the cache bandwidth becomes also relevant for the execution time of these techniques. The techniques were adapted to multi-core processors and now several successful implementations exist [2, 5, 10, 12]. In this study we use the particularly simple *cache accurate time skewing (CATS)* algorithm that is not much longer than the naive implementation, see Alg. 1.

The contribution of this paper is to analyze in detail what impact the system and cache bandwidth have on efficient stencil computations. While the naive implementation is known to be memory bound and to scale linearly with the system bandwidth, for the time skewing methods the situation is quite different because cache misses are reduced to such great extent that the cache bandwidth becomes an important performance factor. For this more general situation we develop a performance model, validate it across many processor generations and thus determine how the scaling of the system and cache bandwidth affects hardware performance. These insights are useful for identifying the most performance relevant features of future systems with respect to stencil computations.

The next section discusses our hardware and software setup for the performance and cache analysis. Section 3 compares the execution times of the naive and CATS for varying problem sizes. In Section 4 we vary the cache size and explain how CATS can predict the performance on virtual machines with different cache sizes. Based on the previous data, the performance model is developed in Section 5 and Section 6 validates it and estimates performance for new hardware configurations. We draw conclusions in Section 7.

2 Test Environment

2.1 Hardware Setup

Table 1 lists the configuration of our hardware. We refer to the machines by the processor name throughout the paper. The first two represent the older generation of single-core, multi-CPU workstations. The last two offer four cores either in one or two sockets and feature integrated memory controllers. The Core i7 940 is also used to simulate a Core i5 dual-core system by executing only two threads on this processor and this configuration is labeled Core i5 Sim.

The second half of the table presents results of synthetic benchmarks. The 'number of threads' row shows how many threads were used during the computation. This number is fixed on each machine. The bandwidth benchmarks have been performed with the RAMspeed benchmarking tool with SSE reads. All x86-64 capable machines run 64-bit Linux and the GNU C++ 4.3 compiler. On the Xeon MP machine we use 32-bit Linux and the GNU C++ 4.2.1 compiler.

The measured system floating point performance numbers come from our own benchmarks. The peak performance value (see Table 1) is the maximum number of independent, alternating multiply and add instructions executed per second on all available SSE units. This gives us the maximum overall system performance. However, as our performance model assumes faster computation than data fetching from the cache (the problem is cache bandwidth bound in the cache), we are more interested in the application specific performance of the stencil computation. Therefore, we implement the 7-point stencil computation on registers as a series of accumulations of products and present the results as measured stencil DP FLOPS in Table 1. This value is lower than the measured peak performance because of the dependency (read-after-write) between the instructions in the pipeline.

2.2 Software Setup

We use two schemes for the tests with iterative stencil computations. The naive scheme consists of perfectly nested for-loops that traverse the entire spatial domain and the outer most loop that repeats the stencil application multiple times (NaiveSSE_3D() in Alg. 1). The cache accurate time skewing (CATS) scheme exploits temporal locality between the consecutive iterations of the stencil. It also consists of nested for-loops and allows similar parallelization and vectorization as the naive scheme, only there are more loops and they appear in different order(CATS_3D() in Alg. 1). Both schemes are vectorized with SSE2 intrinsics on the inner most loop and parallelized using pthreads on the outer most loop using the number of threads according to Table 1.

All our performance tests share certain properties:

- computation in double precision,
- ping-pong iterations with two vectors,
- constant general 7-point stencil in 3D (7 multiplications plus 6 additions) with Dirichlet boundary condition,
- 3D domains ranging from 0.5 to 128 million elements, corresponding to 8MiB-2GiB of data.

We have also tested some other configurations with single precision, variable stencils (banded matrix), in-place stencil updates, 1D and 2D domains and obtain qualitatively similar relations as discussed below although the quantitative results can vary significantly with the parameters. In the following, we prefer to deliver a consistent analysis from start to end for the specified parameters rather than jumping between different parameter configurations.

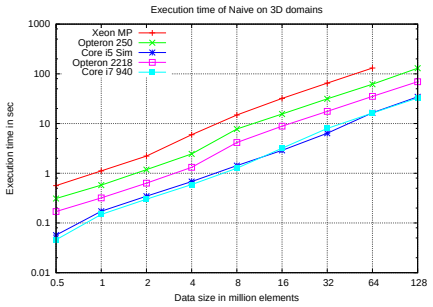

Figure 1. Execution time of the naive scheme for varying domain sizes in 3D.

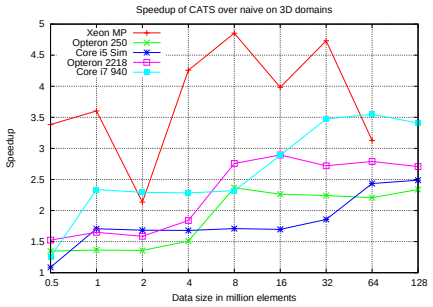

Figure 2. Speedup of the CATS scheme against the naive scheme for varying domain sizes in 3D.

In Section 4 we simulate cache misses of the naive and CATS scheme. The cache miss analysis is performed using the cachegrind profiler from the valgrind 3.2.1 tool suite. We simulate a processor with one cache level and interpret recorded read and write misses as the misses of the last level cache of our machines. Because cachegrind does not simulate multi-threaded programs realistically, we record the cache misses separately for each thread on its piece of the domain and sum up the values. This can lead to slightly higher values because of some additional data reuse in the shared L3 cache, but Section 4 shows that for CATS this is not a problem because there is hardly any additional data reuse apart from the explicitly accounted for reuse.

3 Naive and Time Skewed Stencil Computations

The naive scheme implementation progresses with the entire domain one timestep after the other. As the domain size is usually bigger than the cache, each pass thrashes the cache contents entirely. This makes the naive scheme depend mainly on the system bandwidth for performance. Figure 1 shows the linear relation between the domain size and execution time. The only noticeable non-linearity can be observed for the Core machines at the transition from 0.5 to 1 million elements. This jump is caused by the large L3 cache size when two 0.5 million vectors fit completely into the cache, but the two 1.0 million vectors do not.

Figure 2 shows the speedup of the CATS scheme over the naive implementation. For moderately sized domains the naive implementation benefits from the cache capacity and reuses data elements from neighboring 2D slices. But the speedup increases in a step when neighboring 2D slices do not fit into the cache any more. As the Core i5 Sim machine runs with 4MiB of cache memory per thread, this increase happens only at 64 million elements, which is much later than in the case of any other platform.

The maximum speedup value is closely related to the ratio of the last level cache bandwidth to the system bandwidth (see Table 1). Systems with larger discrepancy between bandwidths benefit more from the increased temporal locality of the computation in CATS, effectively generating larger speedups. An example of such a system is the Xeon MP which has the largest ratio of bandwidths namely 5.4 (Table 1). Accordingly, for 32 million elements, CATS achieves a speedup of almost 5 times. For the same domain size the Core i7 940 with a bandwidth ratio of 3.6 accelerates by a 3.5 factor. A similar result is observed for the Opteron 2218 machine. The smaller speedups on the last two systems are not surprising because of their integrated memory controllers and the dual/triple channel memory interfaces providing up to 6 times more memory bandwidth than the Xeon MP.

Figures 3 and 4 show the execution times of the CATS scheme. The algorithm requires prior knowledge about the available cache size per thread. We use 1024KiB and 128KiB

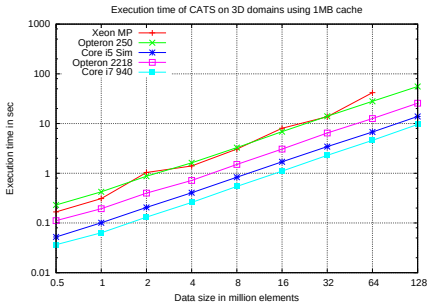

Figure 3. Execution time of the CATS scheme for varying domain sizes in 3D with cache size parameter 1024KiB.

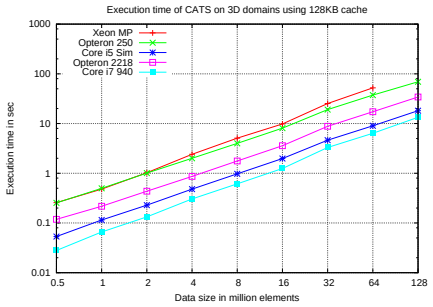

Figure 4. Execution time of the CATS scheme for varying domain sizes in 3D with cache size parameter 128KiB.

for the cache size settings, respectively. For both configurations CATS shows consistently faster execution time than the naive scheme. Previously noticed non-linear performance scaling at the transition from 0.5 to 1 million elements on the Core machines is no longer visible. The CATS scheme scales consistently with the domain size no matter if the entire domain fits into the cache or not. Clearly, in the case of the 128KiB cache size, CATS exploits temporal localities less efficiently compared to the 1MiB setting, resulting in increased execution time. But the 128KiB configuration of CATS is still noticeably faster than the naive implementation, e.g. for 32 million elements, the speedup is 2.6x on the Xeon MP compared to 4.7x for the 1MiB cache setting. On machines with smaller cache to system memory bandwidth ratio like Core i5 or Opteron 2218, the speedup is less impressive, but the execution times are still half that of the naive approach.

For the naive scheme the Opteron 250 is faster than Xeon MP (see Figure 1) by a factor of 1.8x. As the algorithm is memory bound the difference in timings directly relates to the difference in the measured system bandwidths: 6.4 GB/s for the Opteron 250 vs. 3.2 GB/s for the Xeon MP (see Table 1). However, for the CATS scheme, cache bandwidth is the main performance limiting factor. For the Opteron 250 and the Xeon MP, the cache bandwidth is approximately equal and therefore both machines perform similarly on CATS.

A comparison of the Core systems (see Figures 3 and 4) leads to similar conclusions. Both systems are equal in terms of system bandwidth, effectively achieving the same results for the naive approach. However, for the CATS scheme the quad-core Core i7 940 performs significantly better. The difference comes from the increased aggregated cache bandwidth when all four cores are in use. In general, one can say that the performance order of the machines in Figure 1 reflects the ranking in system bandwidth, while their order in Figure 3 reflects the ranking in cache bandwidth.

4 Varying Cache Size

The previous section already showed two different plots of CATS performance depending on the passed cache size parameter: Figure 3 for $Z = 1024$KiB and Figure 4 for $Z = 128$KiB. In this section we want to fix the problem size to either a 200^3 domain (8 million elements) or a 400^3 domain (64 million elements) and look at the performance scaling on machines with different cache sizes. Figures 5 and 6 show the execution times for varying cache sizes. This will be used in the next section to derive a performance model. But first we need to explain why these numbers predict the performance of CATS on hardware configurations with smaller cache sizes, although the actual machines on which we execute have obviously a fixed hardware cache size.

The reason is an invariance property of the CATS scheme. Given a cache size param-

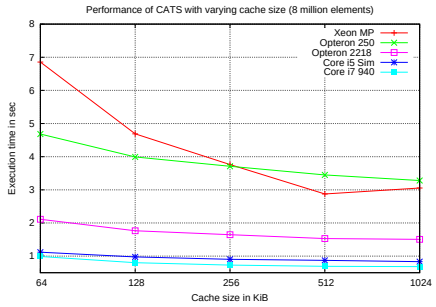

Figure 5. Execution time of CATS for a 200^3 domain and varying cache sizes.

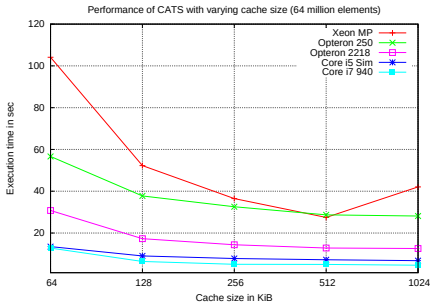

Figure 6. Execution time of CATS for a 400^3 domain and varying cache sizes.

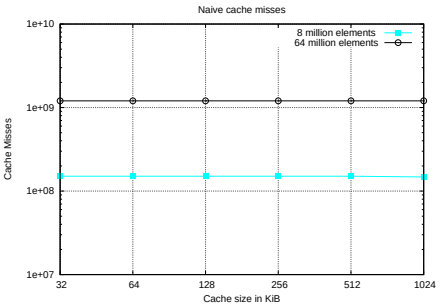

Figure 7. Cache misses of the naive scheme for a 200^3 and a 400^3 domain and varying cache sizes.

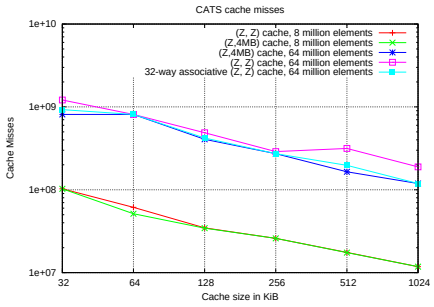

Figure 8. Cache misses of the CATS scheme for a 200^3 and a 400^3 domain and varying cache sizes.

eter Z it will incur the same number of cache misses on a machine with Z cache, $2 \cdot Z$ or even $4 \cdot Z$ cache. It only matters if the actually available cache size is bigger than or equal to the specified parameter. The CATS scheme optimizes the entire computation very carefully with respect to the given cache size parameter, so even if the actual cache is bigger there will be hardly any additional savings on cache misses.

Figure 8 confirms the above reasoning. If CATS is fed with the same Z value, then the cache misses on a machine with 4MiB cache size (CATS $(Z, 4\text{MiB})$) are only insignificantly lower than on a machine with Z cache (CATS (Z, Z)). Only in case of the large 400^3 domain (128 million elements), we obtain a discrepancy due to imperfect cache associativity. Increasing the cache associativity from 8-way to 32-way recovers the almost identical behavior.

This invariance of CATS is very useful, because the arithmetic intensity of stencil computations is very low, so their performance depends mainly on the number of cache misses and the system and cache bandwidth of the machine. So if the cache misses do not change when we run on a machine with much larger cache than the cache size parameter, then the performance should not change either. Practically, by setting the cache size parameter Z to some value, e.g. 128KiB, we obtain the execution time of a virtual machine with this cache size $Z = 128\text{KiB}$, even though the actual execution takes place on a machine with 4MiB cache size. Figure 8 clearly shows that even such big difference of 4MiB to 128 KiB has almost no impact on the number of incurred cache misses and thus the performance of CATS.

Something similar holds trivially for the naive scheme as demonstrated in Figure 7. In this case the number of cache misses is almost constant no matter how big the cache is, because the entire domain is fetched for each iteration of the stencil, and the entire domain is always larger than the cache size, so no data reuse between the stencil iterations can

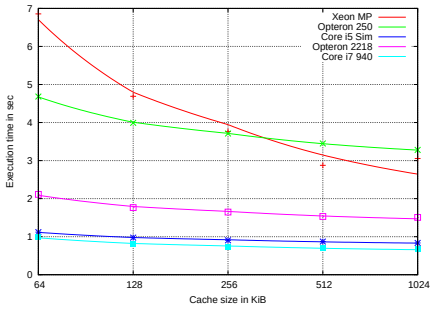

Figure 9. Execution time of CATS for a 200^3 domain and varying cache sizes. The points show the measured execution time, while the lines show our fitted performance model based on the number of cache misses.

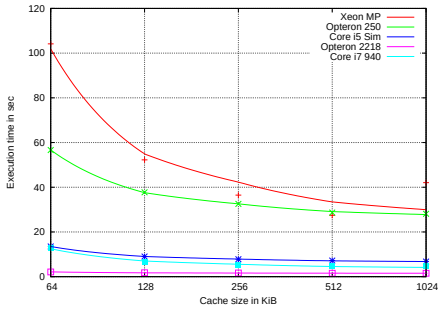

Figure 10. Execution time of CATS for a 400^3 domain and varying cache sizes. The points show the measured execution time, while the lines show our fitted performance model based on the number of cache misses.

occur. Comparing Figures 7 and 8, we see that CATS produces fewer cache misses even for small cache sizes. For growing cache sizes, cache misses incurred by CATS decrease rapidly, producing less than a tenth of the naive cache misses for a 1024KiB cache size.

5 Performance Model

In the previous section we explained how we can use CATS to estimate the execution time on the same machine where we virtually vary the size of the available cache. In this section we derive a performance model that links the number of simulated cache misses directly to the measured execution time. The model estimates the execution time $E(m)$ as

$$E(m) := C_l \cdot (m_r(Z) + m_w(Z))/b_{\text{sys}} + (C_d TNN_s - C_l \cdot m_r(Z))/b_{\text{cache}}, \quad (1)$$

where $m_r(Z)$ and $m_w(Z)$ are the simulated numbers of read and write cache misses for varying cache size Z, $C_l = 128$B is the size of the cache line, $C_d = 8$B is the size of a domain element (double precision), T is the number of iterative stencil applications, N is the number of elements in the domain, and $N_s = 7$ is the number of non-zero stencil weights. The model has two free parameters which are the system bandwidth b_{sys} and the cache bandwidth b_{cache}. The parameters are estimated by a least-square-fitting of the model to the measured execution times from Figures 5 and 6.

The first addend in Eq. 1 contributes with the time necessary for the transfer from main memory due to the cache misses. The second addend corresponds to the time of all remaining transfers (there are TNN_s double computations) from the cache to the processing units. Figures 9 and 10 show the measured execution times as points and the fitted performance models as line plots.

The model considers only a two level memory hierarchy: the system memory level where the domain resides and a cache level in which the temporal locality is exploited. Real machines have multiple cache levels and so the effect of the temporal locality is higher or smaller depending on which cache level it occurs in. So the model does not capture the secondary effects of higher level caches. However, the cache bandwidth ratios on-chip are clearly smaller than the bandwidth ratios between the last level cache and the system bandwidth (Table 1) and the schemes do not use any explicit optimization for high level caches, so that the secondary effects are less relevant.

Previous comparison of the execution time of the naive scheme in Figure 1 against CATS in Figure 4 reveals that even on a machine with just 128KiB cache, CATS performs

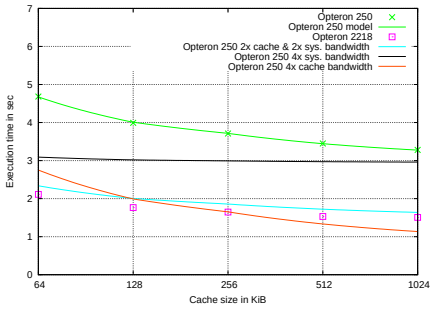

Figure 11. Performance evaluation of the Opteron 250 against the Opteron 2218 w.r.t the CATS scheme.

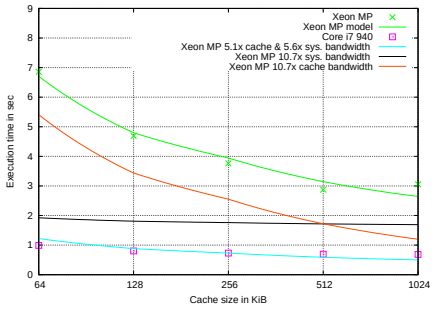

Figure 12. Performance evaluation of the Xeon MP against the Core i7 940 w.r.t the CATS scheme.

clearly better. Looking at the cache misses in Figures 7 and 8, we see that at 128KiB CATS has already a dramatic reduction of cache misses against the naive implementation. For larger caches the difference in cache misses continues to grow at the same pace, but Figures 5 and 6 show diminishing performance returns from the cache miss reduction after 128KiB. Why has the cache miss reduction at first a strong impact on performance while later this impact is much smaller? This behavior can be understood from Eq. 1 by computing the speedup obtained from halving the cache misses:

$$\begin{aligned} \frac{E(2m)}{E(m)} &= 1 + \frac{C_l \cdot ((m_r + m_w)/b_{\text{sys}} - m_r/b_{\text{cache}})}{E(m)} \geq 1 + \frac{C_l(m_r + m_w)C_b}{E(m)} \\ C_b &:= (1/b_{\text{sys}} - 1/b_{\text{cache}}) > 0\,. \end{aligned} \tag{2}$$

At first we see that the speedup depends directly on the discrepancy between the system and cache bandwidth encoded in C_b. There is also a second effect; in the beginning when the data traffic produced by the cache misses $C_l(m_r + m_w)$ is a significant fraction of the overall traffic $C_d TNN_s$, i.e. $\frac{C_l(m_r+m_w)}{E(m)} \approx 1$, the speedup is high. But once this fraction becomes small, i.e. $\frac{C_l(m_r+m_w)}{E(m)} \ll 1$, the speedup becomes negligible. We have a strong scaling model similar to Amdahl's Law: even many-fold reductions on a small fraction of the overall execution time give only small absolute returns. This explains the bended curves of measured execution time in Figures 5 and 6 despite the linear decrease in cache misses from Figure 8. The performance model formalizes this behavior and fits the bended curves closely to the measured execution times as shown in Figures 9 and 10

Now that we understand which parameters control the achieved speedup, we can use the performance model to estimate the impact of changes in these parameters.

6 Model Evaluation

With our parameter controlled performance model, we can roughly predict the effects of increasing the system bandwidth or increasing the cache bandwidth on the execution time, and thus evaluate the impact of these system parameters on the performance of stencil computations. First we want to use this feature to validate the model by increasing the parameters of our older test machines, such as to reach the execution times of the newer machines. In a second step, we increase the parameters even further predicting the performance of non-existent hardware.

Figure 11 compares the Opteron 250 with the Opteron 2218, while Figure 12 looks at the Xeon MP and the Core i7 940. The points in the figure denote the actual execution times while the lines denote the model. From Table 1 we compute the ratio between the benchmarked system and cache bandwidths of Opteron 250 and Opteron 2218, we do

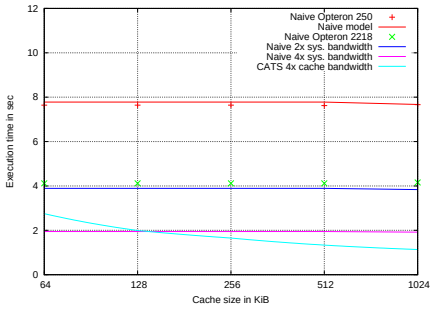

Figure 13. Performance evaluation of the Opteron 250 against the Opteron 2218: A comparison between the naive and the CATS schemes.

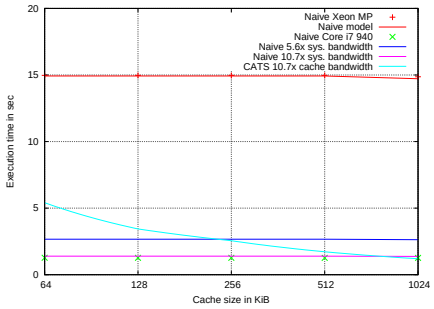

Figure 14. Performance evaluation of the Xeon MP against the Core i7 940: A comparison between the naive and the CATS schemes.

the same for the Xeon MP and the Core i7 940, and use these ratios to scale our model parameters. We model three scenarios: scaling b_{sys}, scaling b_{cache}, and scaling both.

If the model is accurate then the scaling with the above ratios of the system and cache bandwidth in the models of the old machines should recover the measured execution times of the new machines. In fact, the Opteron 250 model with doubled cache and system bandwidth comes close to the measured performance of the Opteron 2218 in Figure 11 and the Xeon MP model with 5.1x cache and 5.6x system bandwidth scaling comes close to the measured performance of the Core i7 940 in Figure 12. This validates our assumption that for stencil computations the cache and system bandwidth parameters matter most and the actual processor architecture is rather irrelevant, even though the Xeon MP design is eight years older and completely different from the Core i7 940 design. Because of this successful validation, we are confident to use this model also for new parameter configurations of non-existent hardware.

On one end of the spectrum, scaling the system bandwidth of the Opteron 250 by a factor of 4 in Figure 11 does not benefit the CATS scheme so much. It rather flattens the curve making it similar to the naive performance, because such a great increase in system bandwidth would put it on par with the cache bandwidth. On the other end of the spectrum, a quadrupled cache bandwidth in Opteron 250, accelerates the CATS scheme even beyond the Opteron 2218 performance on large cache sizes. This result is obtained by only changing the cache bandwidth in the CPU, the system bandwidth would still be half that of the Opteron 2218 system. We see that the performance of the CATS scheme reacts very favorably to cache bandwidth scaling even if it is not accompanied by a faster system bus. This is a very cost-efficient way of increasing the overall performance, although it deteriorates the ratio of off-chip to on-chip bandwidth which is usually blamed for bad performance of stencil computations. Instead, we see that performance depends strongly on the implementation of stencil computations, worsening this ratio can actually be a good thing to do.

The usual advocation of multi-channel memory buses simply comes from the fact that most stencil computations are implemented in a naive way that depends on the system bandwidth for performance. Figure 13 shows that doubling or quadrupling the system bandwidth accelerates the naive scheme by almost the same factor. But changing the system bandwidth so radically is a very expensive procedure. In comparison, we see that the inexpensive quadrupled cache bandwidth on CATS still outperforms the enhanced naive scheme by factor 1.8x if the cache size is 1024KiB.

In Figure 12, we perform a similar analysis for the old Xeon MP and the new Core i7 940 Intel architectures. The benchmarked system and cache bandwidth ratios between them are 5.6x and 5.1x, respectively (see Table 1). We use these factors to scale the Xeon MP performance model. The predicted CATS performance with $5.6x$ system and 5.1x

cache bandwidth is in fact almost the same as the measured execution times on the Core i7 940. Moreover, we see in Figure 12 that further doubling the system bandwidth but leaving the cache bandwidth on the original value would not get us this far. On the other hand, if we leave the very low system bandwidth of the Xeon MP intact, and increase its cache bandwidth to twice that of the Core i7 940, we would still fall short of the Core i7 940 performance but would already beat the much more expensive system bandwidth scaling by the same factor for the 1024KiB cache size.

The situation for the naive scheme on the Xeon MP in Figure 14 is very similar to the AMD equivalent from Figure 13. The naive schemes benefit proportionally from the system bandwidth scaling; however, for the 1024KiB cache size, the far more inexpensive multiplication of cores in the Xeon MP without changes to the system bandwidth would already deliver superior results.

The discussed relation of system parameters for CATS clearly supports the option of increasing the cache bandwidth rather than the system bandwidth. In current systems cache bandwidth increases automatically with the growing number of cores provided that each core has its own locally connected cache. This scaling option comes with the overhead of keeping a large number of caches coherent; however, CATS features big tiles and requires data synchronization only at their boundaries if they are processed by different threads. Therefore, this synchronization could be performed explicitly with little overhead on a system with non-coherent caches. By further increasing the speed of the local caches, one could quickly obtain enormous speedups in stencil computations using time skewing schemes. One may even reduce the cache size in favor of more cache bandwidth if the discrepancy to the system bandwidth is not too high. If the cache to system bandwidth discrepancy becomes very high the CATS performance curves become very steep and give bad results for small cache sizes, see Figure 12.

Unfortunately, the cost-efficient strategy of deteriorating the ratio of off-chip to on-chip bandwidth through the introduction of faster caches does not help the naive codes. So we are in a dilemma here. Using clever schemes we can increase the performance of stencil computations radically by the simple scaling of the aggregate cache bandwidth, but all naive codes would suffer in this situation and even more severely demand an increase in system bandwidth. Therefore, concerning iterative stencil computations, the bandwidth wall problem is only partially a hardware issue, more importantly we have a software issue of ineffective implementations in many codes. The expensive scaling of the system bandwidth through multi-channel memory interfaces could stop without deteriorating performance if more codes would change the way iterations of stencil computations are implemented. Of course, not all applications can benefit from time skewing, so one would need to know the fraction of iterative stencil computations in the application mix to determine which amount of system and cache bandwidth would give the best performance per cost ratio on average.

7 Conclusions

We have examined the impact of system and cache bandwidth on the naive and the cache accurate time skewing (CATS) scheme for iterative stencil computations. The schemes exhibit almost completely opposite behavior. While the naive scheme requires high system bandwidth for performance, the same stencil computation can be performed with a time skewing scheme much faster if only the cache bandwidth in the CPU is increased. The latter option gives by far the more cost-efficient performance gains, e.g. we could execute on the ten years old Xeon MP as fast as on a Core i7 940 if only sufficient cache bandwidth in the Xeon MP were provided without the need for any improvement of its outdated system bus. So the paradoxical conclusion is that for iterative stencil computations further deteriorating the ratio of off-chip to on-chip bandwidth is the cheapest way to higher performance. Unfortunately, the situation is more complex in practice because not all stencil computations occur in iterations and many of them operate with varying

rather than constant coefficients which puts additional strain on the system bus. In future, we want to extend the performance model so that it allows to predict the behavior for more computational patterns. However, even the restricted model makes it clear that a solution to the bandwidth wall problem should not be sought solely in system bandwidth scaling, because it is not necessarily the limiting factor even if the data is much bigger than the caches and has to be accessed many times.

References

[1] M. Awasthi, K. Sudan, R. Balasubramonian, and J. Carter. Dynamic hardware-assisted software-controlled page placement to manage capacity allocation and sharing within large caches. In *IEEE International Symposium on High Performance Computer Architecture*, pages 250–261, 2009.

[2] M. M. Baskaran, A. Hartono, S. Tavarageri, T. Henretty, J. Ramanujam, and P. Sadayappan. Parametrized tiling revisited. In *Proc. of the International Symposium on Code Generation and Optimization (CGO'10)*, 2010.

[3] A. Hartstein, V. Srinivasan, T. Puzak, and P. Emma. On the nature of cache miss behavior: Is it $\sqrt{2}$? *Journal of Instruction-Level Parallelism*, 10, 2008.

[4] G. H. Loh. 3d-stacked memory architectures for multi-core processors. In *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*, pages 453–464, Washington, DC, USA, 2008. IEEE Computer Society.

[5] A. Nguyen, N. Satish, J. Chhugani, C. Kim, and P. Dubey. 3.5-D blocking optimization for stencil computations on modern CPUs and GPUs. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–13, 2010.

[6] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin. Scaling the bandwidth wall: challenges in and avenues for CMP scaling. *SIGARCH Comput. Archit. News*, 37(3):371–382, 2009.

[7] S. Somogyi, T. F. Wenisch, A. Ailamaki, and B. Falsafi. Spatio-temporal memory streaming. In *ISCA*, pages 69–80, 2009.

[8] Y. Song and Z. Li. New tiling techniques to improve cache temporal locality. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, 1999.

[9] R. Strzodka, M. Shaheen, and D. Pajak. Time skewing made simple. In *Proceedings ACM symposium on principles and practice of parallel programming*, PPoPP '11, Feb. 2011.

[10] R. Strzodka, M. Shaheen, D. Pajak, and H.-P. Seidel. Cache oblivious parallelograms in iterative stencil computations. In *ICS '10: Proceedings of the 24th ACM International Conference on Supercomputing*, pages 49–59. ACM, 2010.

[11] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen. A novel architecture of the 3D stacked MRAM L2 cache for CMPs. In *HPCA*, pages 239–249, 2009.

[12] M. Wittmann, G. Hager, and G. Wellein. Multicore-aware parallel temporal blocking of stencil codes for shared and distributed memory. In *Proc. Workshop on Large-Scale Parallel Processing (LSPP'10) at IPDPS'10*, 2010.

[13] M. Wolf. More iteration space tiling. In *Proceedings of Supercomputing '89*, 1989.

[14] D. Wonnacott. Using time skewing to eliminate idle time due to memory bandwidth and network limitations. In *Proceedings of International Parallel and Distributed Processing Symposium*, 2000.