Experiences with the Sparse Matrix-Vector Multiplication on a Many-core Processor

Juan C. Pichel and Francisco F. Rivera
Centro de Investigación en Tecnoloxías da Información (CITIUS)
Universidade de Santiago de Compostela (Spain)
juancarlos.pichel,ff.rivera@usc.es

***Abstract*—Industry is moving towards many-core processors, which are expected to consist of tens or even hundreds of cores. One of these future processors is the 48-core experimental processor Single-Chip Cloud Computer (SCC). The SCC was created by Intel Labs as a platform for many-core research. The characteristics of this system turns it into a big challenge for researchers in order to extract performance from such complex architecture.**

In this work we study and explore the behavior of an irregular application such as the Sparse Matrix-Vector multiplication (SpMV) on the SCC processor. An evaluation in terms of performance and power efficiency is provided. Our experiments give some key insights that can serve as guidelines for the understanding and optimization of the SpMV kernel on this architecture. Furthermore, a comparison of the SCC processor with several leading multicore processors and GPUs is performed.

***Keywords*-many-core; sparse matrix; performance; power efficiency;**

I. Introduction

Nowadays modern processors contain a reduced number of cores, typically from 2 to 12. The general trend in the industry is to include a higher number of cores on the same chip, the so-called *many-core* processors. These future processors are expected to include tens or even hundreds of cores. In this paper we consider one of those experimental many-core processors, the Single-Chip Cloud Computer (SCC). The SCC is a 48-core experimental processor created by Intel Labs as a platform for many-core software research.

The objective of this work is to study and explore the behavior of irregular applications on the SCC processor. As representative example we have selected the Sparse Matrix-Vector product (SpMV), which is one of the most important computational kernels in scientific and engineering applications. It is notorious for sustaining low fractions of peak performance (typically, about 10%) on modern processors. Among the factors that have an impact on its performance, two of them are of the utmost importance: the memory bandwidth limitation and the low locality caused by the irregular and indirect memory accesses. An additional drawback is that the sparse matrix patterns that characterize such irregular accesses are only known at run-time.

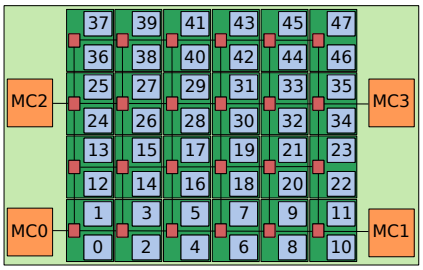
The SCC processor consists of 48 cores connected through a mesh network, four memory controllers given access to the main memory and several power management functions. These characteristics turn the SCC processor into an ideal candidate to give researchers key insights about the behavior and performance of the SpMV on the future commercial many-core processors. In this work we present an in-depth analysis of the effects on the SpMV performance of several issues. This is the case of the influence of mapping the units of execution to different cores. We have also analyzed the relation between performance and working set size, and the effect of the irregular accesses. Furthermore, since the SCC processor allows to change the cores, mesh network and memory clock frequency, an evaluation in terms of performance and power efficiency of different configurations has been provided. Finally, an architectural comparison with several leading multicore processors and GPUs was performed.

This paper is structured as follows: Section II introduces a description of the SCC processor. Section III presents the main characteristics of the SpMV kernel together with the sparse matrices testbed. The results of the experimental evaluation on the SCC processor are analyzed and discussed in Section IV. Section V discusses relevant previous work. The paper finishes with the main conclusions extracted from this work.

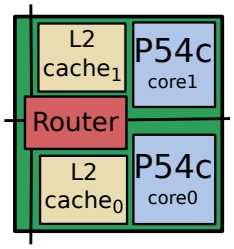
II. The Single-Chip Cloud Computer (SCC)

The Single-Chip Cloud Computer (SCC) is a 48-core experimental processor created by Intel Labs as a platform for many-core software research. The SCC consists of 48 independent x86 cores arranged in 24 tiles. Figure 1 provides an overview of the SCC processor together with a scheme of a single tile.

A tile consists of two P54c Pentium cores with an increased L1 instruction and data cache sizes of 16 KB each [1]. It also holds separate 256 KB L2 caches for each core. Caches are 4-way set associative with a pseudo-LRU replacement policy. L2 cache is write-back only. The SCC offers no coherency among cores caches to the programmer. This coherency must be implemented through software methods, by flushing caches for instance. Furthermore each



(a)



(b)

Figure 1. Overview of the SCC processor (a) and a tile scheme (b).

tile also contains a 16KB (8 KB per core) Message Passing Buffer (MPB). It is intended to support a message-passing programming model on the chip. The MPB is implemented as shared memory that can be accessed efficiently by all 48 cores on the SCC. Furthermore, each tile has its own frequency domain in such a way that the clock frequency can be set separately for every of the 24 tiles. Clock frequencies from 100 to 800 MHz can be used.

The mesh network is a simple 2D grid that connects all tiles [2]. Each tile contains a router with four ports, which connects its tile with other units on the SCC in four different directions. The routing scheme is a simple (x,y) scheme, that is, data is routed first horizontally and then vertically through the network. The network has its own clock and power source and can be initialized to operate on either 800 MHz or 1.6 GHz. However dynamic changes during runtime are currently not supported by the chip.

The overall system admits a maximum of 64GB of main memory accessible through 4 DDR3 memory controllers evenly distributed around the mesh. Our setup uses 32 GB. Each core is attributed a private domain in this main memory whose size depends on the total memory available (640 MB in the system used here). The memory controllers are attached to the routers of four tiles at the edges of the chip at the coordinates (0,0), (2,0), (0,5) and (2,5) (see MC boxes in Figure 1(a)). Six tiles (12 cores) share one of the four memory controllers to access their private memory. In addition, a part of the main memory is shared between all cores. The memory controllers operate on their own clock and power source, and can be clocked at 800 or 1066 MHz. But, like the mesh, the clock frequency of the memory controllers must be chosen during the initialization of the chip.

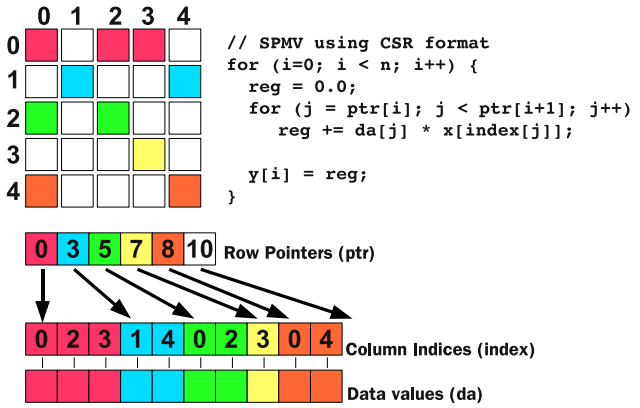


Figure 2. Compressed-Sparse-Row (CSR) format example, and a basic CRS-based sparse matrix-vector product (SpMV) implementation.

Intel provides RCCE (pronounced “rocky”) [3], a simple message passing library which supports basic message passing functions that are specifically designed to use the special architecture characteristics of the SCC (e.g. the message passing buffer). RCCE has two basic communication primitives: point-to-point communications and collective operations. It also provides access to other entities like the voltage controller or it allows the allocation of shared memory. When using RCCE, the cores are not directly accessed by their identification (as shown in Figure 1(a)). Instead, for each execution of a RCCE application, the cores to be used and their order can be configured differently [4]. Each one of the participant cores is mapped to an unit of execution (UE). Within the applications, these UEs are addressed by their ranks, in a similar way to addressing MPI processes.

III. Sparse Matrix-Vector Multiplication (SpMV)

In this work the sparse matrix-vector product (SpMV) operation is considered. This kernel is notorious for sustaining low fractions of peak processor performance due to its indirect and irregular memory access patterns. Let us consider the operation `y=A×x`, where `x` and `y` are dense vectors, and `A` is a $n \times m$ sparse matrix. The most common data structure used to store a sparse matrix for SpMV computations is the Compressed-Sparse-Row format (CSR). According to this format the nonzero elements (nnz) of a sparse matrix with n rows are stored contiguously in memory in row-major order. The `index` array stores the column indices of each element in the original matrix, and the `ptr` array of size $n+1$ stores a pointer to the beginning of each row. da stores the nnz data values of the sparse matrix.

Figure 2 shows, in addition to an example of the CSR format for a 5×5 sparse matrix, a basic implementation of

Table I
MATRIX BENCHMARK SUITE.

	Matrix	# rows (n)	# nonzeros (nnz)	nnz/n	ws (MBytes)
1	TSOPF_FS_b300_c2	56814	8767466	154.3	101.4
2	F2	71505	5294285	74.0	62.0
3	ship001	34920	4644230	133.0	53.8
4	thread	29736	4470048	150.3	51.7
5	gupta2	62064	4248286	68.5	49.8
6	nd3k	9000	3279690	364.4	37.7
7	sme3Dc	42930	3148656	73.3	36.9
8	pct20stif	52329	2698463	51.6	31.9
9	tsyl201	20685	2454957	118.7	28.5
10	exdata_1	6001	2269501	378.2	26.1
11	mixtank_new	29957	1995041	66.6	23.4
12	crystk03	24696	1751178	70.9	20.5
13	av41092	41092	1683902	70.9	20.1
14	sparsine	50000	1548988	31.0	18.7
15	nc5	19652	1499816	76.3	17.5
16	syn12000a	12000	1436806	119.7	16.7
17	li	22695	1350309	59.5	15.9
18	msc10848	10848	1229778	113.4	14.3
19	gyro_k	17361	1021159	58.8	12.0
20	sme3Da	12504	874887	70.0	10.3
21	fp	7548	848553	112.4	9.9
22	e40r0100	17281	552562	32.0	6.7
23	psmigr_1	3140	543162	173.0	6.3
24	rajat15	37261	443573	11.9	5.8
25	ncvxbqp1	50000	349968	7.0	5.0
26	nmos3	18588	386594	20.8	4.8
27	net25	9520	401200	42.1	4.8
28	garon2	13535	390607	28.9	4.7
29	bcsstm36	23052	320606	13.9	4.1
30	Na5	5832	305630	52.4	3.6
31	tandem_vtx	18454	253350	13.7	3.3
32	lhr10	10672	232633	21.8	2.9

the SpMV for this format. This implementation enumerates the stored elements of `A` by streaming both `index` and `da` with unit-stride, and loads and stores each element of `y` only once. However, `x` is accessed indirectly, and unless we can inspect `index` at run-time, it is difficult or impossible to reuse the elements of `x` explicitly. Note that the locality of the accesses to `x` depends on the sparsity pattern of the considered matrix.

As matrix test set 32 square sparse matrices from different real problems that represent a variety of nonzero patterns were selected. These matrices are from the University of Florida Sparse Matrix Collection (UFL) [5]. Table I summarizes the features of the matrices: number of rows/columns (n), nonzeros (nnz), nonzeros per row/colum (nnz/n) and size of the working set (ws). Considering 32-bit integer indexing and double precision arithmetic, the working set in bytes can be calculated as: $4 \times ((n+1) + nnz) + 8 \times (nnz + 2 \times n)$.

IV. EXPERIMENTAL EVALUATION

The SCC platform is based on Ubuntu Linux with a 2.6.32-24 kernel. Codes in this work were written in C and compiled with the Intel's 8.1 Linux C compiler (`icc`). RCCE library was used to parallelize the sparse matrix-vector multiplication code of Figure 2. In particular, the results of this section were obtained by using RCCE 1.0.13 and the compiler optimization flag `-O2`. The partitioning scheme splits the matrix row-wise in such a way that the same amount of nonzeros would be assigned to each unit of execution.

For all the experiments (with the exception of those in Sections IV-D and IV-E), cores, routers and memory are clocked at the default speeds of 533 MHz, 800 MHz and 800 MHz respectively. Note that since the SCC cores do not provide a constant clock that is independent of their operational frequency, `RCCE_wtime()` was used as time measurement function, which returns the RCCE wall time and ensures correct time measurements even for different clock frequencies. In order to measure the performance, we have used the floating operations per second (FLOPS/s), which was calculated by dividing the total number of floating point operations ($2 \times nnz$) by the execution time.

As mentioned, the SpMV kernel is notorious for sustaining low fractions of peak processor performance. Among the factors that have an impact on its performance, two are of the utmost importance: the memory bandwidth limitation and the low locality caused by the irregular and indirect memory accesses. Therefore, it is important to evaluate the influence of the SCC memory hierarchy on the SpMV performance.

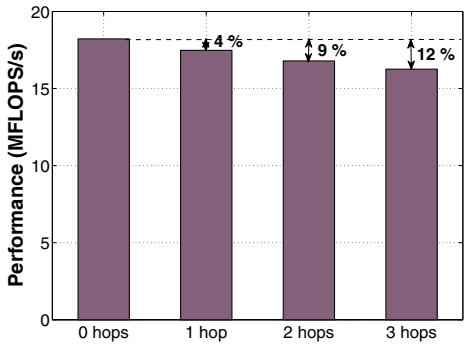
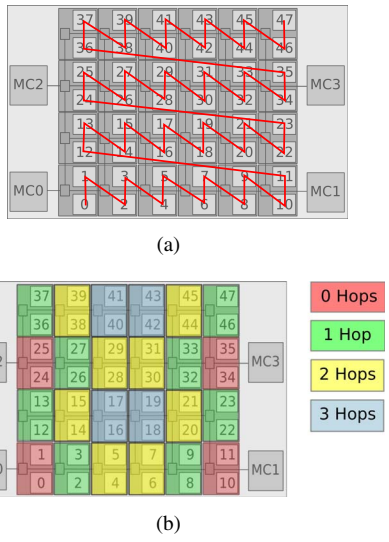


Figure 3. Comparison of the performance obtained when mapping one unit of execution to cores with different distance to their memory module.



(a)

(b)

Figure 4. Different mappings of the units of execution to cores: (a) standard and (b) considering the distance (hops) to the memory module from the core.

Different topics related to this will be analyzed and discussed next in Sections IV-A, IV-B and IV-C. The SCC processor allows to change the cores, mesh and memory clock frequency (see Section II). The impact on the SpMV performance of different processor configurations have been studied in Section IV-D. A power efficiency analysis was also provided. Finally, in Section IV-E, an architectural comparison is shown, in which different multicore processors and GPUs were considered.

A. Mapping Units of Execution to Cores

Next we will evaluate the influence of mapping the units of execution to different cores of the SCC processor. Several topics related to the SCC architecture must be taken into consideration. First, as indicated in Section II, six tiles (12 cores) share one of the four memory controllers to access their private memory. In particular, the cores are, by default,

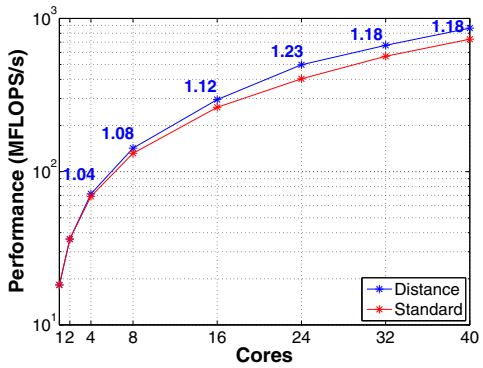


Figure 5. Performance when considering the two mapping configurations for different number of cores.

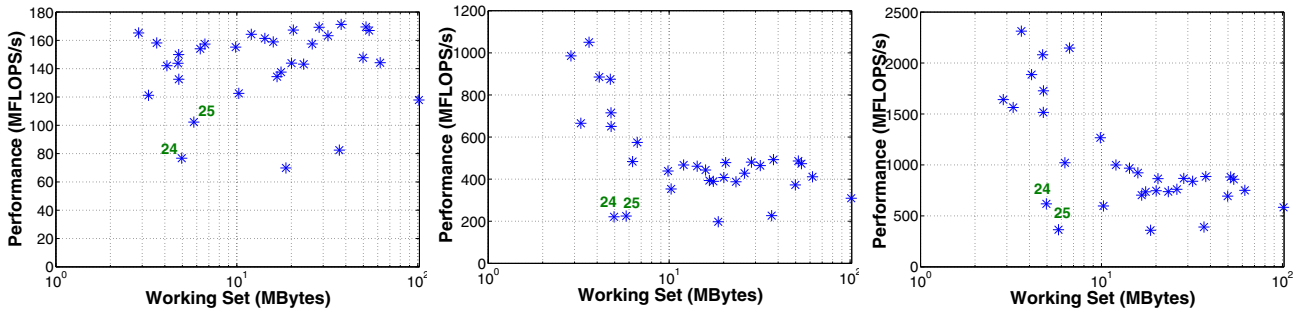
partitioned into quadrants. For example, the lower left quadrant contains cores 0-5 and 12-17 (see Figure 1(a)), and the main memory are accessed through the memory controller `MC0`. And secondly, the memory latency (and bandwidth) of a core depends on the distance to the memory controller [2], [6]. The formula to determine the time required by a core since sending a memory request and receiving the respective data is:

$$40C_{core} + 4 \times n \times 2C_{mesh} + 46C_{mem} \quad (1)$$

where C_{core}, C_{mesh} and C_{mem} are the clock cycles of the core, the mesh network and the memory respectively, and n denotes the number of mesh network *hops* required to reach the memory controller. According to this, the only property that need to be taken into account to measure the memory latency is the number of mesh network hops. For this reason, we will consider the memory performance of cores with 0, 1, 2 and 3 hops to the memory controller, which covers all the possible distances in the default configuration.

Next, we will evaluate the SpMV performance when using a single core accessing the memory. Cores with a different distance (number of hops) to their memory module were analyzed. Results in Figure 3 show the average performance obtained for all the matrices of our testbed. As expected, a better performance is observed when the distance to the memory is the lowest, that is, when the number of hops is 0. According to the Equation 1, this case corresponds with the smallest memory latency. We must highlight that the differences in the performance are noticeable, especially when a 3 hops core is considered. In this case, a degradation of about 12% was observed. This impact can be even more important for parallel executions if the units of execution are not carefully mapped to the cores.

A study to determine the impact on the parallel SpMV performance of different mapping configurations has been carried out. We have compared two mapping configurations: *standard* and *distance reduction*. The standard mapping is the default mapping of units of execution to cores (see



(a) 8 cores (b) 24 cores (c) 48 cores

Figure 6. Performance of the matrices in relation to their working set using different number of cores.

Figure 4(a)). In this case, each RCCE rank is equal to the core identification. For instance, the cores involved in a parallel execution with 4 units of execution would be 0, 1, 2 and 3. This configuration does not take into consideration the distance to the memory. We propose the distance reduction configuration (Figure 4(b)), which maps the units of execution to the available cores with the lowest number of hops to the memory. Considering the previous example, the cores 0, 1, 10 and 11 would perform the SpMV operation when using this configuration.

A comparison of the two mapping configurations is displayed in Figure 5. The graphic shows the average SpMV performance of both mappings for different number of cores. In addition, the speedup values obtained by the distance reduction mapping with respect to the standard configuration are shown. As expected, the distance reduction configuration clearly outperforms the standard one, reaching speedups up to 1.23×. Note that there is no difference in the selected cores by both mappings when using 1 and 2 cores, and therefore, the observed performance is the same.

We conclude that the mapping of units of execution to cores on the SCC has a big impact on the SpMV performance, especially with a high number of cores. The best solution is to select those cores placed near their memory module. For this reason, we have selected the distance reduction mapping configuration in the experiments of the remainder of the paper.

B. Influence of the Working Set Size

Here we will study the influence of the working set of the matrices in the SpMV performance. Figure 6 shows the performance for all the sparse matrices from the testbed relative to their working set size (see Section III for details about its calculation). Results with 8, 24 and 48 cores are displayed.

It is clear from the figure that the performance across the matrix set presents a great diversity. However, we can group the matrices into two categories: matrices whose working set per core fits into the L2 cache of the cores, and those whose working set per core is larger than the L2 cache. In the first case, those matrices will only experience compulsory misses, while in the second case, matrices may also experience capacity ones. When using 8 cores (Figure 6(a)), there are no matrices with a working set per core smaller than the L2 cache. This fact explains the reason why there is not a relation between the working set size and the performance. Nevertheless, when using a higher number of cores, there are many matrices whose working set per core fits into the L2 cache (256 KBytes). Figures 6(b) and 6(c) show the results for executions with 24 and 48 cores respectively. An important boost in the performance of the smallest matrices with respect to the larger ones is observed. For example, considering 24 cores, small matrices reach performance values up to 1 GFLOPS/s, while the bigger ones are always in the range of 450 MFLOPS/s. This behavior is not observed for the small matrices 24 and 25. The cause is the presence of very short row lengths, that is, small nnz/n values. This fact leads to a small trip count in the inner loop of the SpMV kernel, a fact that may degrade the performance due to the increased overhead of the loops [7].

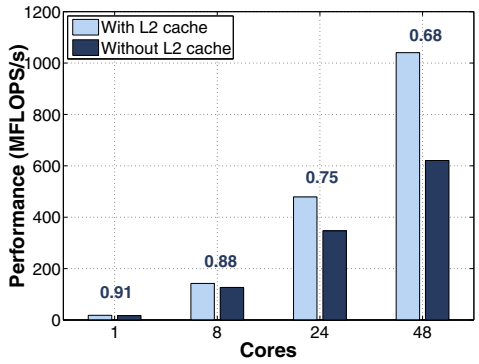


Figure 7. Performance obtained when the L2 caches of the cores are disabled with respect to the default case.

We have performed another experiment to confirm the observations detailed above. The SCC processor allows to

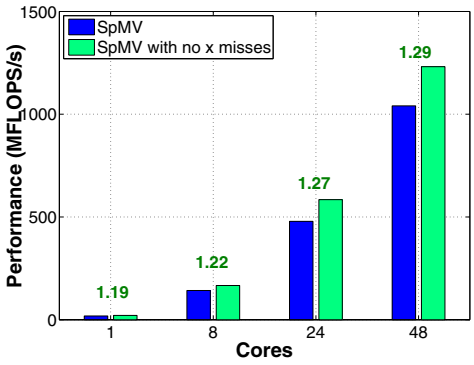


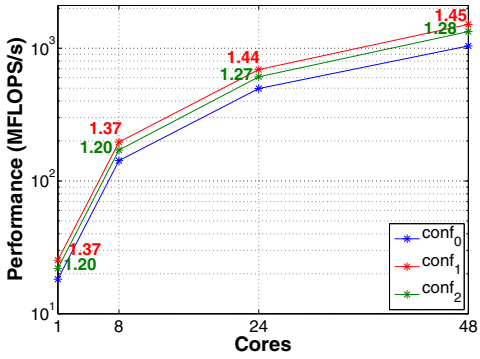
Figure 8. Impact on the performance of the irregular accesses on vector x.

boot the cores disabling the L2 caches. A comparison of the SpMV performance when the L2 caches of the cores are disabled with respect to the default case has been carried out. Results are shown in Figure 7. An important degradation in the performance was obtained, specially as the number of cores grows (for example, 30% when using 48 cores). If the results of Figure 7 with L2 caches disabled are shown by matrix, no relation between the performance and the working set size even for a high number of cores would be found. That is, the behavior would be similar to the one observed in Figure 6(a). These observations confirm that the differences in the SpMV performance between the matrices of the testbed are mainly caused by the capacity cache misses.

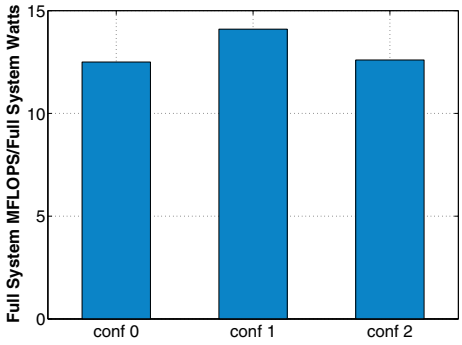
C. Influence of the Irregular Accesses

One of the most important factors that affect the performance of the SpMV is the low locality caused by the irregular and indirect memory accesses. In a similar way to [7], we have performed a test to evaluate the impact of irregular accesses on vector `x` (see the SpMV code in Figure 2). With this purpose a modification of the SpMV code to eliminate cache misses on vector `x` has been carried out. In particular, in this version of the code each reference to `x` will now access only its first element `x[0]`, which results in a perfect access pattern on `x`. In this way, we can assume that any performance variation observed between the original and the modified version of the code is caused by the effect of the irregular accesses on the vector `x`. Figure 8 shows the variations in the performance of both versions of the SpMV code using different number of cores. The average speedup obtained is also provided.

We have observed that the irregular accesses have a big impact on the SpMV performance. Note that a speedup of over 10% has been found in more than 50% of the matrices using different number of cores. This behavior is not observed on other multicore systems [7], where the authors conclude that the irregular accesses is not the



(a)



(b)

Figure 9. Comparison of the performance (a) and power efficiency (b) using different configurations of the SCC processor.

prevailing performance problem. We find the cause in the cache hierarchy of the SCC cores with a small L2 cache of 256 KB. Modern multicore processors have typically three levels of cache, with L3 of bigger size (see Section IV-E). It is worth to mention that the most important speedups using the modified SpMV version were obtained for those matrices that perform poorly with the original SpMV code. For example, this is the case of matrices 24 and 25, whose speedups are higher than 2× (see Figure 6 for the results about the performance of the original SpMV code). Therefore, the influence of the locality in the SpMV performance on the SCC processor is high, especially for those matrices with an irregular nonzero pattern.

D. Evaluation of Different SCC Configurations

The SCC processor allows to change the cores, mesh network and memory clock frequency (see Section II). In particular, each tile (2 cores) has its own frequency domain in such a way that the clock frequency can be set separately for every of the 24 tiles from 100 to 800 MHz. The network can be initialized to operate on either 800 MHz or 1.6 GHz. And the memory controllers can be clocked at 800 or 1066 MHz. The default frequency for the cores, mesh and memory are 533, 800 and 800 MHz respectively. Until now we have

only considered this configuration in our experiments. Now, we will evaluate the impact on the SpMV performance of different configurations of the SCC processor.

We have considered three different configurations, which are defined by the cores, mesh and memory clock frequencies (in MHz):

- $conf_0$ (default): 533, 800, 800
- $conf_1$: 800, 1600, 1066
- $conf_2$: 800, 1600, 800

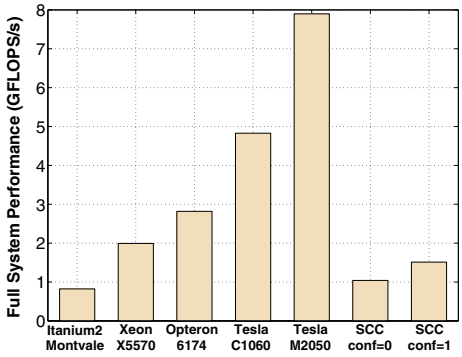
Note that $conf_1$ corresponds with the highest available clock frequency values of the SCC processor, so it is expected to provide the best results in terms of performance. A comparison between the three configurations is shown in Figure 9(a). It is also displayed the speedup obtained by the configurations 1 and 2 with respect to the default one. As expected an important improvement in the SpMV performance has been detected when the clock frequency of the cores, mesh and memory increases. It is especially noticeable with $conf_1$, obtaining a speedup up to $1.45\times$ when using the 48 cores of the SCC. $conf_2$ obtains speedups slightly higher than $1.2\times$. The differences in the performance of $conf_1$ and $conf_2$ are caused only by the clock frequency of the memory, showing $conf_1$ a performance improvement in all cases of about 15%.

However, we have to take into consideration another important issue as consequence of using higher clock frequencies: the power consumption. We have measured the average power consumption of the SCC processor for all the considered configurations when the SpMV kernel is running on. An important increase has been observed for $conf_1$ and $conf_2$ with respect to the default configuration. For example, the SCC processor increases its power consumption from 83.3 to 107.4 W using all the cores and $conf_1$.

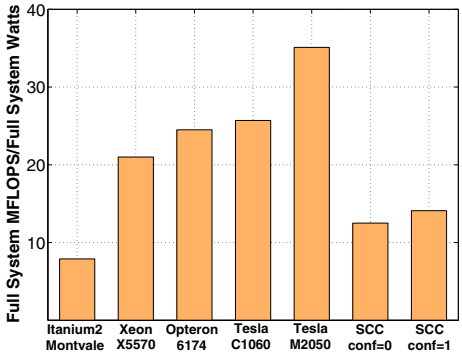
Concerning the power consumption we can study the power efficiency, which is one of today's most important considerations in HPC acquisition. Figure 9(b) shows the MFLOPS/s-to-Watt ratio based on the average SpMV performance and the measured power consumption for the three configurations. Note that the results illustrate the efficiency considering the full system, that is, using the 48 cores of the SCC processor. The best power efficiency is obtained by $conf_1$, demonstrating that the increase in the power consumption (about 30% with respect to $conf_0$) is compensated by a noticeable improvement in the performance. Nevertheless, the efficiency of $conf_0$ and $conf_2$ is practically the same. This fact points out that the performance improvement when using $conf_2$ (Figure 9(a)) is similar to the increase in the power consumption in comparison with the default configuration.

E. Architectural Comparison

Next we will compare the behavior of the SpMV in terms of performance and power efficiency on different systems including the SCC many-core processor. The systems evaluated are the following:



(a)



(b)

Figure 10. Comparison of the performance (a) and power efficiency (b) of different architectures with respect to the SCC processor.

- Intel Itanium2 Montvale: This processor comprises two cores running at 1.6 GHz, and three cache levels (including an unified 9 MB L3 cache per core). The peak performance per core is 6.4 GFLOPS/s. Power consumption: 104 W (TDP).
- Intel Xeon X5570: It is a quad-core processor, which includes three cache levels with a shared 8 MB L3 cache. Each core operates at 2.93 GHz, with 11.72 GFLOPS/s as peak performance per core. Power consumption: 95 W (TDP).
- AMD Opteron 6174: This processor consists of 12 cores running at 2.2 GHz. It has three cache levels with a shared 12 MB L3 cache. Power consumption: 80 W (ADP), 115 W (TDP) [8].
- NVIDIA Tesla C1060: This Graphic Processing Unit (GPU) consists of 240 cores, with a double precision arithmetic peak performance of 78 GFLOPS/s. Power consumption: 187.8 W (TDP).
- NVIDIA Tesla M2050: This GPU was designed using the new Fermi architecture from NVIDIA. It has 448 cores, with a double precision peak performance of

515.2 GFLOPS/s. Power consumption: 225 W (TDP).

Some issues related to these systems must be clarified. The parallelization of the SpMV code of Figure 2 was performed using OpenMP on the Intel and AMD processors. In addition, the Intel's 10.0 Linux C compiler was used with the Xeon and Itanium2 systems, while for the Opteron the compiler was Open64. For the GPUs, we have used the SpMV kernel from the NVIDIA library detailed in [9] and CUDA 3.2. All the results in this section were obtained using the compiler optimization flag `-O2`.

Figure 10(a) shows the average SpMV performance obtained on the systems enumerated above together with the results of the SCC processor using the configurations 0 and 1. These results were calculated using all the available cores on the different systems. Results indicate that SCC only outperforms the dual-core Itanium2 processor. Best results are achieved by the GPUs (especially the one with the Fermi architecture), followed by the Opteron and Xeon processors. We must highlight that the Tesla M2050 obtains a noticeable average performance of 7.9 GFLOPS/s, which means a speedup of 7.6× with respect to the SCC default configuration. This good behavior is caused by the important effort made to improve the double precision arithmetic capabilities of the last generation of NVIDIA GPUs.

Next, we will evaluate the power efficiency of these systems in the same way as above. Figure 10(b) shows the MFLOPS/s-to-Watt ratio based on the average SpMV performance and the power consumption for all the considered systems. The power consumption of the processors has been obtained from the manufacturer's documentation. For comparison purposes, we transform the ACP (Average CPU Power) value provided by AMD for the Opteron processor into its corresponding TDP (Thermal Design Power) according to [8]. Results show that the SCC processor outperforms the Itanium2, but this difference increases with respect to the one observed in Figure 10(a). Overall, the SCC processor behaves better in terms of power efficiency in comparison with the performance results. It is worth to mention that the efficiencies of the Xeon and Opteron processors are quite similar to the observed for Tesla C1060, even when the GPU shows speedups of 2.4× and 1.7× with respect to the performance on both processors. Tesla M2050 is the most power efficiency system considered in this study, reaching a value of 35 MFLOPS/s per Watt.

V. Related Work

The SCC is an experimental processor, but there are some interesting recent works that deal with it. Melot *et al.* [10] evaluated the performance of the SCC processor when accessing the off-chip memory. One of the main conclusions of this work is that, unlike the per-core read memory bandwidth, the write memory access performance decreases when several cores write simultaneously to the memory. In addition, they found that the available bandwidth depends on the memory access pattern, and especially, in the case of irregular accesses. Gschwandtner *et al.* [2] performed an evaluation of the SCC processor using different benchmarks, but they did not consider irregular applications as the SpMV. The authors focus on benchmarking those characteristics of the SCC that previous multi- and many-core processors did not possess. In other work [4], the authors explored the performance of a memory-intensive application on the SCC. They evaluate different mappings of processes to cores as well as different communication scenarios showing significant improvement in terms of performance.

There exist in the literature numerous works that undertake the study and optimization of the SpMV. For example, Williams *et al.* [11] proposed several optimization techniques for the SpMV that are evaluated on different multicore platforms. Some of the proposed techniques are: software pipelining, prefetching approaches, and register and cache blocking. In another paper [7], the authors studied the performance issues of the SpMV on several multicore platforms. Based on different experiments, they gave some useful optimization guidelines about the most important performance bottlenecks and how these should be addressed effectively. In a more recent work [12], the authors studied the behavior of the SpMV on a SMP-NUMA system. They focused on several issues that affect the SpMV performance: the memory and the processor affinity, as well as the influence of the memory hierarchy.

VI. Conclusions

The Single-Chip Cloud Computer (SCC) is a 48-core experimental processor created by Intel Labs as a platform for many-core software research. In this work we have studied the behavior of the Sparse Matrix-Vector product (SpMV) on the SCC processor, which is one of the most important computational kernels in scientific and engineering applications.

We have evaluated the effects on the SpMV performance of several issues. First, we have analyzed the influence of mapping the processes to different cores. The memory latency of the SCC processor depends on the distance of the considered core to its corresponding memory controller. Our experiments show degradations about 12% when using a single core. In the case of parallel executions the impact is even more important. According to this, we propose a mapping configuration that maps the units of execution to the available cores with the lowest distance to the memory. Speedups up to 1.23× using this configuration instead of the standard one were obtained.

The influence of the working set size in the SpMV performance was also studied. The experiments demonstrate that matrices whose working size per core fits into the L2 cache boost their performance when using a high number of cores. For example, these matrices reach performance values up to 1 GFLOPS/s using 24 cores, while the bigger

ones are in the range of 450 MFLOPS/s. The effect of the irregular accesses has also been analyzed. With this purpose a modification of the SpMV code to eliminate cache misses on accesses to vector x has been carried out. Our observations point out that the locality has a big impact on the SpMV performance. Note that a speedup of over 10% has been found in more than 50% of the matrices of our testbed when using the "no x misses" version of the SpMV code. This behavior was not observed on other multicore systems [13].

Since the SCC processor allows to change the cores, mesh network and memory clock frequencies, an evaluation in terms of performance and power efficiency of different configurations has been provided. Speedups up to 1.45× were obtained with respect to the SCC default configuration. Lower differences with respect to the power efficiency has been detected.

Finally, an architectural comparison with several leading multicore processors and GPUs was performed. Results indicate that SCC only outperforms a dual-core Itanium2 system when considering performance and power efficiency. Best results overall were obtained by a Tesla M2050.

ACKNOWLEDGMENT

The authors would like to thank Intel for providing the opportunity to experiment with the many-core processor Single-Chip Cloud Computer (SCC).

This research was partly funded by Ministry of Education and Science (Spain) and Xunta de Galicia through the projects TIN2010-17541 and 09TIC002CT respectively.

REFERENCES

[1] Intel Labs, *SCC External Architecture Specification (EAS)*, november 2010.

[2] P. Gschwandtner, T. Fahringer, and R. Prodan, "Performance analysis and benchmarking of the Intel SCC," in *IEEE Int. Conf. on Cluster Computing (CLUSTER)*, 2011, pp. 139–149.

[3] R. F. van der Wijngaart, T. Mattson, and W. Haas, "Light-weight communications on Intel's single-chip cloud computer processor," *ACM SIGOPS Operating Systems Review*, vol. 45, pp. 73–83, 2011.

[4] M. Korch, T. Rauber, and C. Scholtes, "Memory intensive applications on a many-core processor," in *IEEE Int. Conf. on High Performance Computing and Communications*, 2011, pp. 126–134.

[5] T. Davis, "University of Florida Sparse Matrix Collection," *NA Digest*, vol. 97, no. 23, 1997. [Online]. Available: http://www.cise.ufl.edu/research/sparse/matrices

[6] Intel Labs, *SCC Programmer's Guide*, november 2011.

[7] G. Goumas, K. Kourtis, N. Anastopoulos, V. Karakasis, and N. Koziris, "Performance evaluation of the sparse matrix-vector multiplication on modern architectures," *The Journal of Supercomputing*, vol. 50, no. 1, pp. 36–77, 2009.

[8] Intel, *Measuring Processor Power: TDP vs. ACP*, april 2011, white paper.

[9] N. Bell and M. Garland, "Efficient sparse matrix-vector multiplication on CUDA," NVIDIA, Tech. Rep., 2008.

[10] N. Melot, K. Avdic, C. Kessler, and J. Keller, "Investigation of main memory bandwidth on Intel Single-Chip Cloud Computer," in *3rd Many-core Application Research Community Symposium*, 2011, pp. 107–110.

[11] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, "Optimization of sparse matrix-vector multiply on emerging multicore platforms," in *Proc. of the ACM/IEEE Conf. on Supercomputing*, 2007.

[12] J. C. Pichel, J. A. Lorenzo, D. B. Heras, J. C. Cabaleiro, and T. F. Pena, "Analyzing the execution of sparse matrix-vector product on the Finisterrae SMP-NUMA system," *The Journal of Supercomputing*, vol. 58, no. 2, pp. 195–205, 2011.

[13] G. Goumas, K. Kourtis, N. Anastopoulos, V. Karakasis, and N. Koziris, "Understanding the performance of sparse matrix-vector multiplication," in *Proc. of the Euromicro Conf. on Parallel, Distributed and Network-Based Processing*, 2008, pp. 283–292.

BIOGRAPHIES

Juan C. Pichel received his B.Sc. and M.Sc. in Physics from University of Santiago de Compostela (Spain). In 2006 he received the Ph.D. from University of Santiago de Compostela. He was a visiting postdoctoral researcher at the Computer Architecture and Systems group, University Carlos III de Madrid (Spain), from 2006 to 2008. He also worked as a researcher and project manager at Galicia Supercomputing Center (Spain). Currently he is a post-doc researcher at University of Santiago de Compostela. His research interests include parallel and distributed computing, programming models and software optimization techniques for emerging architectures.

Francisco F. Rivera is a professor of computer architecture at the University of Santiago de Compostela (Spain). He attained his B.Sc. degree in 1986, and his Ph.D. in 1990 at the University of Santiago de Compostela. Throughout his career he has conducted research and published extensively in the areas of computer-based applications, parallel processing and computer architecture. He has conducted international and national projects. He was the director of five PhD thesis. His current research interests include compilation of irregular codes for parallel and distributed systems, and its applications, the analysis and prediction of performance on parallel systems and the design of parallel algorithms of sparse matrix algebra and image processing, Grid computing, memory hierarchy optimizations, GIS, etc.