Breaking the Bandwidth Wall in Chip Multiprocessors

Augusto Vega†, Felipe Cabarcas†‡, Alex Ramírez†, Mateo Valero†
†Barcelona Supercomputing Center and Universitat Politècnica de Catalunya
‡Universidad de Antioquia (Medellín, Colombia)
{augusto.vega,felipe.cabarcas,alex.ramirez,mateo.valero}@bsc.es

Abstract—In throughput-aware CMPs like GPUs and DSPs, software-managed streaming memory systems are an effective way to tolerate high latencies. E.g., the Cell/B.E. incorporates local memories, and data transfers to/from those memories are overlapped with computation using DMAs. In such designs, the latency of the memory system has little impact on performance; instead, memory bandwidth becomes critical. With the increase in the number of cores, conventional DRAMs no longer suffice to satisfy the bandwidth demand. Hence, recent throughput-aware CMPs adopted caches to filter off-chip traffic. However, such caches are optimized for latency, not bandwidth.

This work presents a re-design of the memory system in throughput-aware CMPs. Instead of a traditional latency-aware cache, we propose to spread the address space using fine-grained interleaving all over a shared non-coherent last-level cache (LLC). In this way, on-chip storage is optimally used, with no need to keep coherency. On the memory side, we also propose the use of interleaving across DRAMs but with a much finer granularity than usual page-size approaches.

Our proposal is highly optimized for bandwidth, not latency, by avoiding data replication in the LLC and by using fine-grained address space interleaving in both the LLC and the memory. For a CMP with 128 cores and 64-MB LLC, performance is improved by 21% due to the LLC optimizations and an extra 42% due to the off-chip memory optimizations, for a total 1.7 times performance improvement.

I. INTRODUCTION

In the last years, new CMP designs were introduced in the growing domain of throughput-aware computing [7], [13]: GPUs (e.g., NVIDIA's graphic cards), general-purpose computing on GPUs (as the Intel Larrabee and IBM Cell/B.E.) and DSPs. In such designs, the memory system can be organized based on two models: hardware-managed coherent caches and software-managed streaming memory [11]. In the latter case, the CMP usually incorporates on-chip local memories and data is transferred to and from those memories using direct memory accesses (DMAs). Typical applications executed on throughput-aware CMPs, as real-time computer graphics, video processing, medical-image analysis, molecular dynamics, astrophysical simulation, and gene sequencing [7], are decomposed into data blocks that can be processed massively in parallel. In presence of enough parallelism, DMA transfers can be overlapped with computation to hide latency [17]. Therefore, in throughput-aware CMPs, the latency of the memory system has very little impact on performance and, instead, memory bandwidth becomes critical.

As an example, each Synergistic Processing Element (SPE) in the Cell/B.E. architecture [9] has a 256-KB local memory. Data transferring between this memory and main memory is done using high-bandwidth DMAs. With only 8 such SPEs, the Cell/B.E. already had to resort to a high bandwidth Rambus XDR memory system to satisfy the required bandwidth. The Xbox 360 video game console [1] is another good example of the high bandwidth requirements in throughput-aware CMPs. In this case, the CPU (known as Xenos) is connected to a GDDR3 memory system providing 22.4 GB per second to satisfy the demand of just three cores. Therefore, in such scenarios, the memory system has to be highly optimized for bandwidth.

Furthermore, the increment in the number of cores in current CMPs has posed additional stress to the memory bandwidth [2]. For that reason, recent designs have adopted caches to alleviate the pressure imposed to the memory interface. However, such caches are traditional organizations from the domain of performance-aware computing, optimized for latency rather than bandwidth. For example, the NVIDIA Fermi architecture [5], [13], [14] organizes its 512 cores in 16 clusters of 32 cores each. Inside each cluster there is a local memory that can act as an L1 cache. Unlike previous NVIDIA products (like the G80 and GT200), all clusters share a 768-KB L2 cache to capture temporal data reuse, and reduce off-chip memory traffic.

From our point of view, the adoption of caches optimized for latency in the domain of throughput-aware computing has to be done thoroughly, and its different needs have to be taken into account. Performance-aware computing is latency bounded. Instead, throughput-aware computing is bandwidth bounded. Figure 1(a) shows the impact on performance as a function of the number...
of cores in a throughput-aware CMP (averaged for six scientific kernels, presented in Section IV: Check-
LU, Cholesky, FFT-3D, K-means, K-NN and MatMul),
assuming an ideal 0-cycle latency memory system. As it
can be observed, the bandwidth provided by the memory
system (chart series) has a very significant impact on the
execution time. For instance, in the case of a memory
system providing 25.6 GB/s, increasing the number of
cores from 16 to 128 improves performance by 2.1 times.
However, if the memory system provides 204.8 GB/s,
the execution time is reduced by close to a factor of 6.
Figure 1(b) also shows the impact on performance as a
function of the number of cores in a throughput-aware
CMP (for the same set of applications), but considering
different latencies (chart series), and assuming a high-
bandwidth memory system. In this case, because mem-
ory bandwidth is large enough, the latency of the mem-
ory system has not impact on performance. Therefore,
in throughput-aware scenarios, the memory system has
to be designed to tackle the “bandwidth wall”, instead
of the “latency wall”.

In addition, the number of pins to access off-chip
memory is not growing at the same pace as the band-
width requirements. According to ITRS projections [6],
the pin count just grows about 10% per year. Even
employing high performance memories, that pin count
increase is not enough to satisfy such bandwidth de-
mands. For that reason, it is completely necessary to
adopt caches in order to filter off-chip memory traffic. In
this sense, many points arise regarding its organization:

• Private caches with data replication (to optimize
access latency) vs. shared caches without data repli-
cation (to optimize capacity and bandwidth).

• If data is replicated across caches, the impact of the
data coherence mechanism.

• If data is not replicated, the impact of the address
space interleaving granularity.

Those questions motivate us to present a complete
memory system organization conscientiously-designed
for throughput-aware CMPs. Its most important charac-
teristics are the following:

• Data replication is avoided to improve capacity.

• The address space is interleaved in a fine-grained
fashion across cache blocks and across memory
controllers to improve bandwidth. The finest-
grained interleaving considered in our experiments
is 128 bytes (the LLC line size). In such case,
consecutive LLC lines are spread across different
LLC blocks, as explained in Section V-A.

• Since data is not replicated in the cache, there is no
need to deal with coherence issues.

The key insight of our proposal is that a fine-grain
partitioning of the address space enables higher band-
width, since multiple transfers can proceed in parallel,
at the expense of locality, due to data being spread all
over the available storage. While this trade-off is harmful
for performance-aware architectures, it is beneficial for
throughput-aware CMPs.

We compare our proposal against a memory system
optimized for latency. Both alternatives are evaluated on
a throughput-aware CMP with a master-worker execu-
tion model and support for DMA transfers. For a CMP
with 128 cores and 64-MB last-level cache (LLC), the
partitioned scheme with fine-grained interleaving shows
21% performance improvement, due to 57% higher
cache hit rate and 3.84 times higher bandwidth, over
a traditional LLC with data replication. Additionally,
the fine-grained (128-byte) interleaving across memory
controllers provides an extra 42% improvement on per-
formance. By adopting both optimizations, we obtain a
total 1.7 times performance improvement.

II. RELATED WORK

In [16], Rogers et al. present an analytical model to
study the memory bandwidth as a bottleneck for perfor-
ance scaling in CMPs. Starting from a baseline 8-core
CMP, performance scalability is analyzed for the next four chip generations, according to Moore’s Law. Due to limitations in the number of pins and power, bandwidth does not grow at the same pace as performance. Hence, different memory traffic reduction techniques are considered: cache compression, DRAM and 3D-stacked caches, link compression and sectored caches, among others. As in our work, Rogers et al. also concentrate their study in the bandwidth wall as a bottleneck for performance scalability. However, while they apply an analytical model, we perform a much more detailed analysis of the LLC and off-chip memory organization. For a given LLC area and size, we emphasize the ways to make an optimum use of them (for instance, by interleaving the address space in a fine-grained manner across cache blocks or memory channels). In that sense, our conclusions are usually hard to obtain from analytical models.

In [12], Liu et al. also present an analytical model to study memory bandwidth partitioning and its interaction with LLC partitioning in CMPs. Bandwidth partitioning is implemented using a token bucket algorithm. Each thread sends off-chip requests to a bucket. A token generator distributes tokens between buckets (with rates proportional to the fractions allocated to different threads) and an off-chip request can leave the bucket as far as there exists a corresponding token. Liu et al. model general-purpose CMPs, while we consider throughput-aware scenarios. In their work, threads are implemented as independent applications: there is neither data sharing among them nor coherency traffic. However, we consider that data sharing is a more realistic picture in current chip multiprocessors as well as cache coherency traffic. Furthermore, while Liu et al. model a small 4-core CMP, we consider larger scenarios with up to 128 cores.

In [8], Hardavellas et al. propose Reactive NUCA, a mechanism for LLC data placement in CMPs, targeting both latency and capacity. The design relies on the classification of different cache access patterns in server and multi-programmed applications: shared data is placed at fixed address-interleaved LLC portions (slices), private data is kept in the LLC slice close to the requester and data with a certain sharing degree is replicated across groups of slices. Although there is plenty of works in the literature on data placement for CMP architectures, we just mention Hardavellas et al.’s because it was the latest one at the moment our paper was being written. In their work, Hardavellas et al. consider server and multi-programmed applications, while we concentrate on throughput-oriented workloads that can benefit from streaming memory systems to tolerate high latencies. In that sense, we put our attention in the memory bandwidth, because this is the bottleneck that limits the performance of current throughput-aware CMPs. Because latency is not a problem in our case, our approach is indeed simpler and it does not depend neither on the operating system nor on the application characteristics to achieve significant bandwidth improvements and optimal capacity use. Furthermore, we also consider larger systems with tens or hundreds of processors.

In [10], Kelm et al. propose the Rigel accelerator architecture, which can support over 1000 cores. Rigel groups processors in clusters, and cores within a cluster share a common cache. Clusters are connected and grouped into a tile, and all tiles are attached to a global LLC. Coherence between cluster caches is software managed, and supported via specialized synchronization structures. The Rigel architecture is closely related to the architecture we consider in our work: processors, grouped in clusters, share common local caches, and all the clusters are interconnected together and to a global LLC. However, in Rigel, data replication is allowed across cluster caches, making it necessary to support cache coherency. It is implemented at software level, adding responsibility to the application. In our architecture, however, there is no need to keep coherence, which means no coherence traffic and no need to deal with coherence issues. Although Rigel and our architecture are similar, we can not make a deep comparison between them due to the lack of detailed information about Rigel.

III. THROUGHPUT-AWARE CMP ARCHITECTURE

To evaluate our proposal against a memory system optimized for latency, we consider a CMP with a master-worker execution model and support for DMA transfers. The architecture is composed of worker processors (cores) grouped in clusters (Figure 2). Each cluster also contains a local interconnection, and all clusters, together with the memory interface, are connected to a global interconnection. Besides the worker processors, there are a small number of high-performance master processors, which spawn tasks to be executed by workers.

![Fig. 2. Evaluated DMA-based CMP architecture (for sake of simplicity, LLC is not shown in this picture).](image-url)
power in-order CPU, a local memory and a DMA controller. The worker CPU can only access its local memory, which is used for both code and data. All external memory accesses are managed through a programmable DMA controller. Workers program the DMA controller to fetch the required input data for a task. In the same way, when the worker finishes the execution of a task, it programs its DMA controller again to write back the task output data. After task execution, the worker notifies its availability to the master processor in order to receive a new task to be executed. This DMA-based design allows the architecture to fully support double buffering without any interference between execution and data prefetching of subsequent tasks. By overlapping communication and computation through the use of double buffering, it is possible to tolerate high memory latencies.

IV. METHODOLOGY

The LLC schemes considered in this work were modeled using an in-house CMP trace-driven cycle-accurate simulator [15]. CPU modules (masters and workers) are fed with traces of scientific applications written using a task-based programming model. The traces are sequences of tasks that master processors schedule on worker processors. Table I shows a summary of the main architectural parameters simulated.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workers</td>
<td>16, 32, 64, 128 SPE-like cores (8/cluster) @3.2GHz</td>
</tr>
<tr>
<td>Local memory</td>
<td>256 KB (one local memory per worker)</td>
</tr>
<tr>
<td>LLC</td>
<td>4WSA, 128b lines, 1 R/W CPU port, 1 R/W mem. port</td>
</tr>
<tr>
<td>MSHR</td>
<td>64 entries each; one independent MSHR per block</td>
</tr>
<tr>
<td>Buses</td>
<td>8 bytes/cycle bandwidth, bidirectional bus</td>
</tr>
<tr>
<td>DMA engine</td>
<td>Up to 16 concurrent DMAs; 128-byte DMA packages</td>
</tr>
<tr>
<td>Mem. system</td>
<td>4 mem. controllers @25.6GB/s each (102.4GB/s total)</td>
</tr>
</tbody>
</table>

**TABLE I**
SIMULATED ARCHITECTURAL PARAMETERS.

For the experiments in this paper, we use six parallel scientific kernels: Check-LU, Cholesky, FFT-3D, K-means, K-NN and MatMul. These benchmarks were written in the Cell/B.E. variant of the StarSs [3] programming model. The traces collected from these benchmarks contain the information about the required computation time for different phases in the processors as well as the inter-processor communications through DMA transfers.

V. LAST-LEVEL CACHE ORGANIZATION

As stated in Section I, there is a trend to adopt latency-aware cache-based memory systems into the throughput-aware computing domain to alleviate the pressure on bandwidth requirements. Although such CMP architectures can benefit from caches, we should take into account that traditional cache systems are designed to optimize access latency rather than bandwidth. Even in designs with shared LLCs, some kind of data replication is allowed in the cache to improve access latency. In the Intel Larrabee architecture, for instance, the LLC (L2) is divided into subsets. Each core has fast access to its own L2 subset, and data replication is allowed between them. Cache coherence mechanisms are required to keep L2 subsets coherent [18]. As another example of latency-aware LLC in current designs, the IBM Power7™ processor incorporates a 32-MB LLC (L3) shared by its eight cores. The L3 cache is divided into eight 4-MB segments, each one attached to one core. In this way, each core has preferred access latency to its local segment, and data can be replicated into multiple segments to improve access latency to shared lines [22].

Due to this trend to adopt latency-aware cache-based memory systems, in this work we evaluate two alternatives in the context of throughput-aware computing: one optimized for access latency and another optimized for bandwidth and capacity (our proposal).

(a) Latency-aware organization.

(b) Bandwidth-aware organization.

Fig. 3. Evaluated last-level cache alternatives.

In this architecture, the LLC is divided into multiple blocks. Each cluster contains a block connected to its local network (Figure 3). Each block has its
own independent MSHR (Miss Status Holding Register) and two full duplex ports (one for processor requests and responses, and the other for memory requests and responses). For the scheme with data replication, we model an invalidation-based cache coherence protocol: when a CPU modifies a cache line, all the copies on other caches are invalidated. Cache access time has been determined using CACTI [21] for the considered sizes.

In the scheme optimized for latency, the LLC is composed of independent blocks and data can be replicated across them. For instance, in the example shown in Figure 3(a), data A is replicated and always accessed locally. In this way, cores can have nearby copies of data in order to reduce the latency to access the cache. However, having multiple copies of data diminishes the effective capacity of the cache and makes it necessary to implement a coherency mechanism.

In the scheme optimized for bandwidth, the LLC is also composed of a set of independent blocks but neither multiple copies nor block migration between them is allowed. In the example shown in Figure 3(b), there is a single instance of data A and some cores have to access it remotely, thus paying an additional latency. In this scenario, data is statically placed using bit indexing: a set of bits in the data address is used to choose unambiguously the cache block where the data is placed.

A. Interleaving Granularity

In our proposal, the address space is partitioned between LLC blocks based on the data address. For an N-block LLC, we employ $\log_2(N)$ bits to determine the destination block. Taking the $\log_2(N)$ least significant address bits, data placement is interleaved at a fine granularity across blocks. On the other end, taking the $\log_2(N)$ most significant bits, the placement is interleaved with coarse granularity. Also, any point in between can be chosen.

The finest-grained interleaving considered in our experiments is 128 bytes, i.e. the LLC line size. In such case, consecutive LLC lines are spread across different LLC blocks. On the other hand, the coarsest-grained interleaving considered is 32 KB, meaning that the address space is spread across different LLC blocks every 256 LLC lines (lines 0 to 255 are mapped to LLC block 0, lines 256 to 511 are mapped to LLC block 1, and so on).

Figure 4 shows the impact on bandwidth and performance (averaged for all evaluated applications) obtained by the worker processors as a function of the interleaving granularity, ranging from 128 bytes (the baseline) to 32 KB. In each chart, 16, 32, 64 and 128 cores are plotted in a 64-MB LLC configuration. We can observe that the finest-grained interleaving (128 bytes) is the best choice for the considered numbers of cores. In particular, for 128 cores there is an improvement of 10\% on bandwidth and 4\% on performance using a 128-byte interleaving with respect to the typically used 4-KB, and 34\% on bandwidth and 13\% on performance when compared against a 32-KB interleaving. Fine-grained interleaving shows the highest DMA bandwidth and performance because it allows highly-balanced access to the cache blocks. With coarse-grained interleaving, a cache block could be stressed by all processors simultaneously during short periods of time, thus becoming a bottleneck.

For the rest of this work, we adopt the 128-byte fine-grained granularity to interleave the address-space across cache blocks because it shows the highest DMA bandwidth and performance.

B. Partitioned vs. With-Replication Scheme

In this section, we compare a 128-byte interleaved partitioned LLC (that provides the highest DMA bandwidth) against a latency-aware LLC with data replication.

Figure 5(a) shows the relative bandwidth for the partitioned LLC with respect to the organization with data replication, as a function of the number of cores. As
it can be observed, the bandwidth improvement for the case with partitioned LLC significantly increases with the number of cores: for a 64-MB LLC and 64 workers, the bandwidth is 1.5 times better, while for 128 workers it is almost 4 times better. This is because there are more data replication and more coherence invalidations in the LLC optimized for latency, as the number of LLC blocks increases. Figure 5(b) shows the performance speedup for the partitioned LLC with respect to the organization with data replication, for the same experiment. In this case, both alternatives present similar performance for 64 workers or less. But for 128 workers, the scenario with partitioned LLC shows an improvement near 20% with respect to the LLC with data replication. We expect this gap to continue growing for larger numbers of workers, because bandwidth (and not latency) becomes more and more critical as the number of cores increases.

As we could anticipate, the partitioned LLC is better for throughput-aware CMPs. The advantage of replicated caches is the increased locality and lower access latency. However, as we have shown in Figure 1(b), latency is irrelevant in throughput-aware scenarios. In such cases, the partitioned scheme provides higher bandwidth and better off-chip filtering. Figure 6 shows the hit rate for both alternatives, adopting a 64-MB LLC. As it can be observed, the partitioned LLC outperforms the scheme with data replication. Moreover, the hit rate changes slightly as the number of cores increases, while in the LLC with replication the hit rate is significantly affected by the number of cores. This is due to the fact that, as the number of cores increases, there are more LLC blocks and, hence, more data replication. Furthermore, in the scheme with replication, a core may invalidate a cache line that may potentially be useful for other cores. In the same figure, the hit rate on invalidated lines is plotted on top of each bar for the LLC with replication. The number of hits on invalidated lines still present on cache is not negligible, and it can be around 6 ~ 7% of the accesses in some scenarios (16, 32 and 64 cores).

At first glance, the high amount of traffic in the global interconnection appears as the main drawback of a partitioned LLC with fine-grained interleaving: every LLC access from a core will evenly access all the LLC blocks, the local and the remote ones. To evaluate this issue, we measured the number of packets in the global interconnection for both LLC alternatives. For the partitioned LLC, traffic is composed of accesses to remote LLC blocks (due to the partitioning) and requests to memory. For the LLC with replication, besides memory requests, we also considered the accesses to remote shared-lines and invalidation requests for cache coherence purposes. Figure 7(a) shows the traffic ratio for the partitioned LLC with respect to the LLC with replication, as a function of the LLC size. As shown in the chart, the traffic increase is not as high as we could expect if we take into account the benefits obtained in bandwidth and performance. For instance, in the scenario with a 64-MB LLC and 128 cores, the traffic in the global interconnection is 32% higher, but the partitioned cache provides 3.84 times improvement on bandwidth and 21% improvement on performance. Likewise remarkable is the trend: the traffic ratio decreases as the cache size or number of cores grow, that is what we expect to happen in future CMPs.
One of the most important characteristics of our proposal is the off-chip traffic filtering due to a better use of the LLC capacity. Figure 7(b) shows the off-chip traffic ratio for the partitioned LLC with respect to the LLC with replication for the same experiment. As it can be observed, the number of accesses that go out of the chip is substantially reduced and this cut is more noticeable as the number of cores and LLC size are increased. This is a very desirable effect because off-chip memory access is one of the main sources of system power consumption.

In addition, it is possible to observe in Figures 7(a) and 7(b) that the traffic ratio for the partitioned LLC with respect to the LLC with replication drops as the number of cores increases. This is due not to a reduction in the traffic for the partitioned LLC, but to the significant increase in coherence traffic for the LLC with replication.

VI. OFF-CHIP MEMORY ORGANIZATION

To improve the off-chip memory bandwidth, we adopt an approach similar to the one used for the LLC: to access the off-chip memory, we employ more than one memory controller, with several channels each, and the address space is interleaved across channels and memory controllers. Although this is not a novel technique [4], [19], [20], in this work we analyze which is the optimal interleaving granularity to obtain the highest bandwidth. The scenarios evaluated here include a 64-MB LLC with 128-byte interleaving granularity (one LLC block per cluster). The configurations considered cover scenarios with 16, 32, 64 and 128 cores, and 2, 4, 8 and 16 LLC blocks. Figure 8(a) shows the normalized DMA bandwidth obtained by the cores as the memory interleaving granularity is increased from 128 bytes to 32 KB (baseline: 4 KB). As it can be observed for a 128-core CMP, the 128-byte interleaving provides 67% more bandwidth than the 4-KB interleaving, which is the most commonly used in current memory systems; and 238% more bandwidth when compared against the 32-KB one. Similar results can be seen for performance in Figure 8(b). Once again, the finest-grained interleaving granularity presents the best result with a 42% better performance than the 4-KB interleaving, and 63% better performance compared to the 32-KB one.

Similarly to what was observed for last-level caches in Section V, fine-grained interleaving also gets the highest DMA bandwidth when employed in the off-chip memory side, because it allows highly-balanced access to memory controllers and channels. Therefore, we conclude from our experiments that a complete memory system thoroughly optimized at both levels, caches and memory, provides significant improvements on bandwidth and performance for throughput-aware CMPs rather than traditional latency-aware memory organizations.

VII. CONCLUSIONS

Throughput-aware CMPs have rapidly become an important specimen in the multicore ecosystem. In such designs, the increase in the number of cores pushes up the bandwidth demand to access off-chip memory. Recent designs have adopted caches to alleviate the bandwidth requirements. But those caches are conceived to improve latency, not bandwidth.

In this work, we presented a re-design of the memory system targeting throughput-aware computing. Instead of a traditional latency-aware cache, we proposed to spread the address space using fine-grained interleaving all over a shared non-coherent LLC to optimize on-chip storage, with no need to keep coherency. On the memory side, we also proposed the use of interleaving across DRAMs but with a much finer granularity than usual approaches. All these optimizations synergistically provide significant improvements on bandwidth and performance. For a CMP with 128 cores and a 64-MB LLC, our proposal shows a 3.84 times improvement on bandwidth due to the LLC organization, and a 1.67 times improvement due to the DRAM organization (128-byte instead of 4-KB interleaving), which stack together for a total 6.4
times bandwidth improvement. For performance, our proposal shows a 1.21 times improvement due to the LLC organization, and a 1.42 times improvement due to the DRAM organization, which stack together for a total 1.7 times performance improvement.

The trend is also remarkable: bandwidth and performance improvements become more significant with the increase in the number of cores or LLC size. The larger the LLC size, the greater the number of blocks it has to be split into. In a latency-aware LLC, that means more data replication and less efficient use of capacity while, in our proposal, the capacity is not affected by replication and, moreover, cores can access evenly all LLC blocks.

ACKNOWLEDGMENTS

We want to thank Alejandro Rico, Carlos Villavieja, Milan Pavlovic, Toni Quesada and Yoav Etsion for their contributions to the simulator. This research is supported by an FPI research grant (BES-2008-004599) and the Consolider contract number TIN2007-60625 from the Ministry of Science and Innovation of Spain, the European Network of Excellence HIPEAC-2 (ICT-FP7-217068), the ENCORE project (ICT-FP7-248647) and the IBM-BSC MareIncognito project. Cabarcas was also supported by the Program AlBan, the European Union Program of High Level Scholarships for Latin America (scholarship No. E05D058240CO).

REFERENCES