Ramirez, Alex; Cabarcas, Felipe; Juurlink, Ben; Álvarez-Mesa, Mauricio; Sanchez, Friman; Azevedo, Arnaldo; Meenderinck, Cor; Ciobanu, Cătălin; Isaza, Sebastian; Gaydadjiev, Georgi

The SARC Architecture

Ramirez, Alex; Cabarcas, Felipe; Juurlink, Ben; Álvarez-Mesa, Mauricio; Sanchez, Friman; Azevedo, Arnaldo; Meenderinck, Cor; Ciobanu, Cătălin; Isaza, Sebastian; Gaydadjiev, Georgi (2010). The SARC Architecture. IEEE Micro, 30(5), 16–29. https://doi.org/10.1109/mm.2010.79

The SARC Architecture

Ramirez, Alex[1]; Cabarcas, Felipe[1]; Juurlink, Ben[2]; Álvarez-Mesa, Mauricio[3]; Sanchez, Friman[3]; Azevedo, Arnaldo[4]; Meenderinck, Cor[4]; Ciobanu, Cătălin[4]; Isaza, Sebastian[4]; Gaydadjiev, Georgi[4]

[1] Barcelona Supercomputing Center

[2] Technische Universität Berlin

[3] Universitat Politècnica de Catalunya

[4] Delft University of Technology

Abstract: The SARC architecture is composed of multiple processor types and a set of user-managed direct memory access (DMA) engines that let the runtime scheduler overlap data transfer and computation. The runtime system automatically allocates tasks on the heterogeneous cores and schedules the data transfers through the DMA engines. SARC's programming model supports various highly parallel applications, with matching support from specialized accelerator processors.

On-chip parallel computation shows great promise for scaling raw processing performance within a given power bud-get. However, chip multiprocessors (CMPs) often struggle with programmability and scalability issues such as cache coherency and off-chip memory bandwidth and latency.

Programming a multiprocessor system not only requires the programmer to discover parallelism in the application, it also requires mapping threads to processors, distributing data to optimize locality, scheduling data transfers to hide latencies, and so on. These programmability issues translate to a difficulty in generating sufficient computational work to keep all on-chip processing units busy. This issue is attributable to the use of inadequate parallel programming abstractions and the lack of runtime support to manage and exploit parallelism.

The SARC architecture is based on a heterogeneous set of processors managed at runtime in a master-worker mode. Runtime management software detects and exploits task-level parallelism across multiple workers, similarly to how an out-of-order superscalar processor dynamically detects instruction-level parallelism (ILP) to exploit multiple functional units. SARC's runtime ability to schedule data transfers ahead of time allows applications to tolerate long memory latencies. We thus focus the design on providing sufficient bandwidth to feed data to all workers. Performance evaluations using a set of applications from the multimedia, bioinformatics, and scientific domains (see the “Target Applications” sidebar for a description of these applications) demonstrate the SARC architecture's potential for a broad range of parallel computing scenarios, and its performance scalability to hundreds of on-chip processors.

The SARC architecture targets a new class of task-based data-flow programming models that includes StarSs, [1] Cilk, [2] RapidMind, [3] Sequoia, [4] and OpenMP 3.0.[5] These programming models let programmers write efficient parallel programs by identifying candidate functions to be off-loaded to worker processors. StarSs also allows annotating the task input and output operands, thereby enabling the runtime system to reason about intertask data dependencies when scheduling tasks and data transfers.

Programming Model

StarSs, the programming model used in this article, consists of a source-to-source compiler and a supporting runtime library. The compiler translates C code, with annotations of the task's inputs and outputs, into a common C code with calls to the supporting runtime library. We chose a software runtime manager to avoid tying the architecture to a particular programming model and its runtime system.

In StarSs, the runtime system manages both data and task scheduling, which do not require explicit programmer intervention. This is similar in spirit to out-of-order processors that automatically detect data dependencies among multiple instructions, build the dynamic data-flow graph, and dispatch instructions to multiple functional units. However, in this case, the data-flow graph is not bounded by the instruction window, the granularity of instructions is much larger, and it does not require in-order commit to support precise exceptions.

An Asymmetric Chip Multiprocessor

Figure 1 shows a logical view of the SARC architecture. It is an asymmetric CMP that includes a few high-performance master processors and clusters of worker processors that are customized to a target application domain. For example, the SARC instance for the H.264 advanced video codec features different accelerator processors for the context-adaptive binary arithmetic coding (CABAC) (entropy decoding) and the macroblock decoding (inverse discrete cosine transform [IDCT], motion compensation, deblocking filter, and so on).

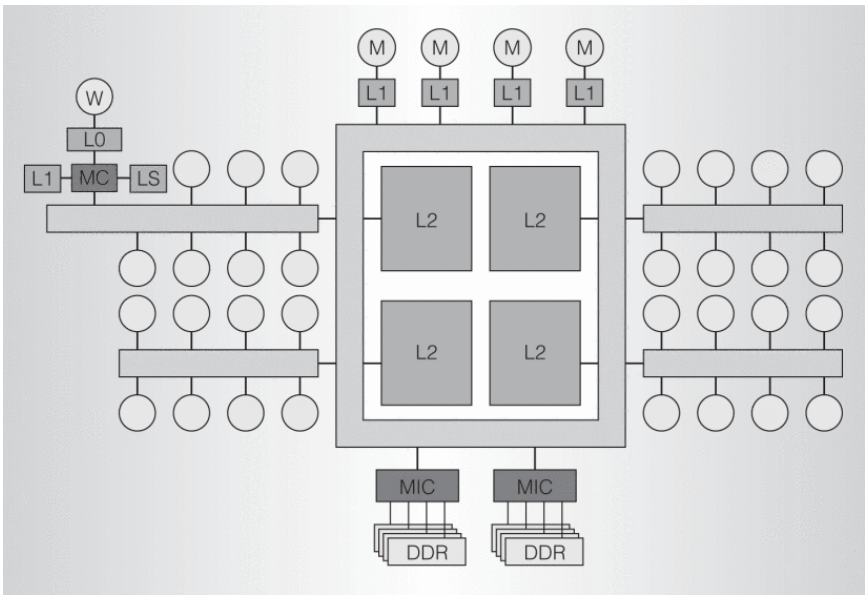

Figure 1. Schematic of the SARC architecture. The number of masters, workers, level-2 (L2) blocks, and memory interface controllers is implementation dependent, as is their on-chip layout.

Master processors

Master processors execute the master threads. They are responsible for starting up the application at the program's main() subroutine. From there, the application can spawn multiple parallel threads that will be allocated to other master processors. Because these processors' main functionality is to sequentially spawn tasks for the workers to execute, their single-threaded performance is critical to the system as a whole. They therefore have a high-performance out-of-order design.

Because they run the only threads whose data-access patterns are not known to the runtime system (the master thread inputs and outputs are not annotated), the masters only access memory through the cache hierarchy. All masters have coherent level-one (L1) instruction and data caches that rely on the replacement policy and the coherency protocol to exploit locality.

Worker processors

Workers off-load task execution from the masters. In addition to the regular cache hierarchy provided to the masters, workers feature a local scratchpad memory. The scratchpads are mapped into the application's logical address space, and are accessed through regular load/store instructions. This means that memory accesses from a worker must go through the translation look-aside buffer (TLB) in their memory controller to be steered toward their scratchpad, a remote scratchpad on another worker, or through the cache hierarchy to the off-chip memory.

To avoid the latency penalty involved in sequential TLB and scratchpad/cache accesses, workers first check a logically indexed and tagged write-through L0 cache. In addition, the L0 cache behaves like a vector cache, and allows unaligned vector load/store operations.[6] An unaligned L0 access can potentially cause two cache misses: one for each half line. Both L0 misses will be resolved by two properly aligned LI accesses. Because LI and L2 caches will only service L0 misses, they do not need to support such unaligned accesses, thus improving their efficiency.

To avoid coherency problems between the distributed scratchpad memories and the cache hierarchy, the LI caches in both masters and workers can only capture addresses in the DRAM physical range. That is, the memory addresses mapped to any of the scratchpad memories are not cacheable.

In addition, each worker features a DMA controller that allows the runtime to overlap data transfer and computation. The DMA controller can copy data from the local memory to off-chip memory or to a remote local memory (and vice versa). Details on the implementation virtualization, transfer between scratchpads, and the DMA interface are available elsewhere.[7]

Target Applications

Our evaluations used four widely used applications representative of different domains: the H.264/advanced video coding (AVC) decoder from the multimedia domain, the FASTA Smith-Waterman protein sequence alignment from the bioinformatics domain, and the dense matrix multiply kernel and Cholesky decomposition from the scientific domain.

H.264/AVC decoder

We developed two parallel versions of the FFmpeg H.264 decoder. Both versions are based on macroblock-level parallelism, which is more scalable than other parallelization approaches such as slice-level or frame-level parallelism.[1]

The first version, referred to as the 2D-wave, is based on the work of Van der Tol et al.,[2] and exploits intraframe macroblock-level parallelism. Each macroblock depends on the reference area in the reference frames and neighboring macroblocks. This leads to a diagonal wave of parallelism progressing through the frame. The second version, referred to as the 3D-wave, exploits both intraframe and interframe macroblock-level parallelism. The 30-wave is based on the observation that interframe dependencies have a limited spatial range because motion vectors are typically small. It is therefore possible to start decoding the next frame as soon as the reference macroblock has been decoded, even if the decoding of the current frame is not finished. This strategy increases the amount of available parallelism significantly beyond what the 2D-wave provides, without increasing the decode latency of individual frames.[3]

Smith-Waterman

The protein sequence alignment problem has multiple levels of parallelism. Most frequently exploited is the embarrassingly parallel situation in which a collection of query sequences must be aligned to a database of candidate sequences.[4] In the SARC project, we developed a parallelization strategy to address the problem of aligning only one query sequence to a single candidate based on the FASTA Smith-Waterman code.[5] The most time-consuming part of the algorithm computes the sum of the diagonals in a dynamically generated matrix, which leads to a 2D wavefront parallelism similar to the one for the H.264 decoder.

The amount of parallelism depends on the size of the compared sequences. A pair of sequences of 4 M and 30 K symbols provides sufficient parallelism to keep 256 processors busy processing blocks of 16 K elements. Longer sequences, such as full genomes, provide sufficient parallelism for even more cores.

Matrix multiply

Matrix multiplication is a well-known parallel problem that has been heavily optimized for many architectures. We start from a blocked algorithm and spawn each block multiplication as a task. Each block of row x column multiplications builds a dependency chain, but there are many independent dependency chains to exploit parallelism. We chose matrix multiply because it is a well-known problem that lets us analyze our architecture under predictable circumstances. It also puts the highest pressure on the memory architecture.

Cholesky decomposition

The Cholesky decomposition, or Cholesky triangle, is a decomposition of a symmetric, positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose. As Figure A shows, the blocked version of the Cholesky decomposition results in an irregular intertask dependency pattern. However, coding such a dependency pattern in the SARC programming model is fairly simple because the runtime library dynamically builds the dependency graph, so it dynamically detects and exploits the parallelism. The available parallelism in Cholesky depends on the graph's maximum width, and diminishes as the algorithm progresses. As we show in the main article, this limits the application's scalability.

References

[1] C.H.Meenderinck et al., "Parallel Scalability of VideoDecoders," J. Signal Processing Systems, vol. 57, no. 2, 2008, pp. 173-194.

[2] E.B. van der Tol, E.G. Jaspers, and R.H. Gelderblom, "Mapping of H.264 Decoding on a Multiprocessor Architecture," Proc. SPIE Conf. Image and Video Comm. and Processing, Int'l Soc. for Optical Eng., vol. 5022, 2003, pp. 707-718.

[3] A. Azevedo et al., "A Highly Scalable Parallel Implementation of H.264," Trans. High-Performance Embedded Architectures and Compilers (HiPEAC), vol. 4, no. 2, 2009.

[4] F. Sánchez Castaño, A. Ramirez, and M. Valero, "Quantitative Analysis of Sequence Alignment Applications on Multiprocessor Architectures," Proc. 6th ACM Conf. Computing Frontiers, 2009, pp. 61-70.

[5] W.R. Pearson, "Searching Protein Sequence Libraries: Comparison of the Sensitivity and Selectivity of the Smith-Waterman and FASTA Algorithms, Genomics, vol. 11, no. 3, 1991, pp. 635-650.

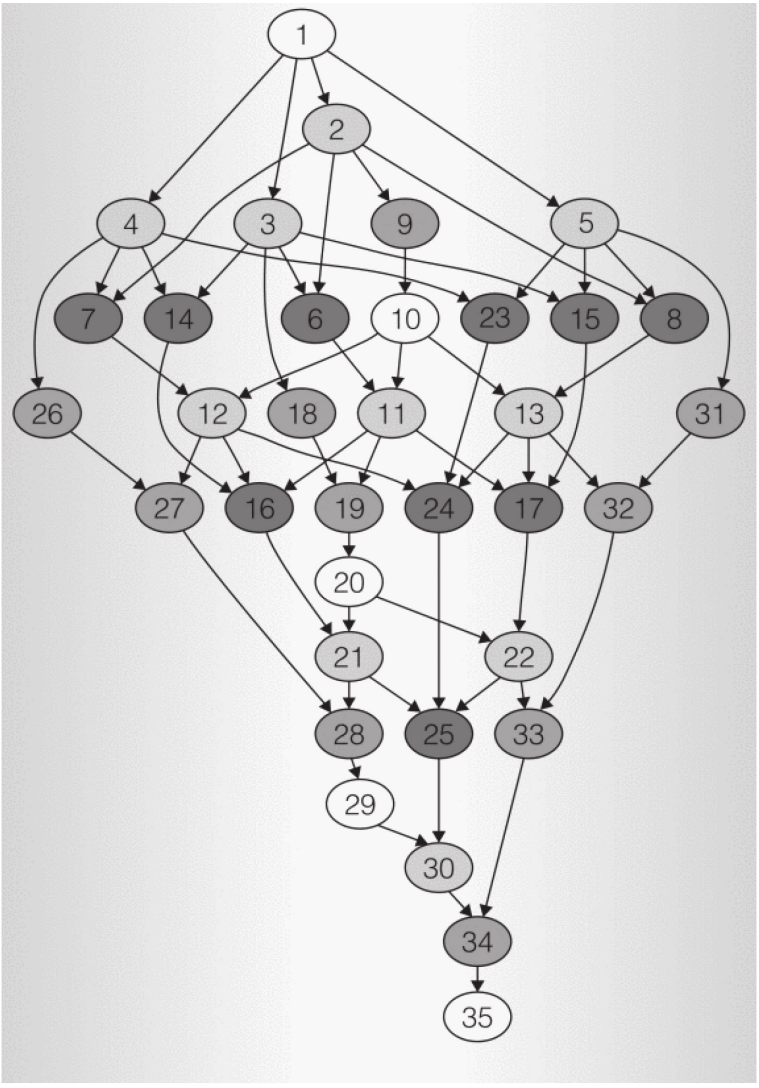

Figure A. Intertask dependency graph of a small 6 x 6 Cholesky decomposition

Shared L2 Cache

All off-chip memory traffic goes through a distributed (or banked) shared L2 cache that captures both misses from the L1 caches and DMA transfers to and from off-chip memory. The L2 cache's distributed structure eliminates the need to maintain coherency across L2 blocks, because a datum is mapped to a particular bank based on its physical address. In addition, the cache structure enables the use of fine-grained interleaving to increase cache bandwidth on consecutive address access. Because the architecture relies on DMAs to transfer data to and from workers, the L2 cache typically encounters coordinated accesses to multiple cache lines. Fine-grained interleaving lets the cache serve multiple parts of a single DMA request in parallel, and increases the effective bandwidth experienced by the request.

The cache's distributed nature leads to a nonuniform cache access time. However, as we show later, the architecture handles long (and variable) latencies without impacting performance. Thanks to the runtime management of data transfers, applications for the SARC architecture can exploit the distributed cache's size and bandwidth benefits without suffering any of the latency penalties.

Because the local memory addresses are not cached on the LI, the L2 cache only needs to maintain coherency with the LI caches. Such a coherency engine is simplified because:

- the shared L2 cache is inclusive of all L1 caches, thus directory state is kept in L2, and is only needed per L2 line; and
- the directory only keeps per-cluster presence bits (not per-LI bits), and

invalidations are broadcast inside each concerned cluster.

Memory interface controllers

The on-chip memory interface controllers (MICs) connect the chip to the off-chip DRAM modules. Each MIC supports several DRAM channels. Internally, each MIC maintains a per-channel request queue, and uses a simple first-in, first-out (FIFO) scheduler to issue requests to the DRAMs. Therefore, requests to a given channel are handled in order, but they can execute out of order with respect to requests sent to another channel.

Given that the MIC will interleave requests from many worker processors, the DRAM bank page buffer will likely not be reused for two consecutive requests. For this reason we use a closed-page DRAM policy.[8]

Similar to the shared cache design, the global address space is interleaved across the different MICs in a fine-grained manner. Given the bulk nature of memory accesses caused by the common use of DMA transfers, such a fine-grained interleaving provides better memory bandwidth because it parallelizes a typical DMA transfer both across MICs and across channels inside each MIC.

Network on chip

A scalable architecture such as SARC must be capable of connecting hundreds of on-chip components. We used the hierarchical K-bus organization for our simulations. A K-bus is a collection of buses. When a node wants to transmit something through the network, it requests permission from the K-bus arbitrator. If there is no previous request for communication with the same destination port, the node is dynamically assigned one of the buses. For example, a 4-bus can accept up to four simultaneous data transfers in a given cycle, as long as no two have the same destination port.

As Figure 1 shows, we organized SARC workers in clusters of eight processors. Each cluster uses a 2-bus for its intracluster network. The cluster has a single (full-duplex) port connecting it to the global interconnect. In a 256-worker SARC configuration, the global NoC connects 32 clusters, 16 masters, 32 cache blocks, and two memory controllers: a total of 82 nodes, not counting the 1/0 controllers. For the purpose of this article, we used a 16-bus for the global NoC.

Sarc Accelerators

The SARC worker processors are based on different designs, depending on the target application domain.

Media accelerator

The SARC media accelerator (SARC-MA) is an application-specific instruction set processor (ASIP) based on the Cell synergistic processor element (SPE). Our goal with this design was to show that adding a few (at most a dozen) application-specific instructions can achieve significant (more than a factor of 2) performance improvements. We chose the Cell SPE as the baseline because it is already optimized for computation-intensive applications but not specifically for H.264 video decoding.

To select the application-specific instructions, we thoroughly analyzed the H.264 macroblock-decoding kernels. We then added 14 instructions, some of which can process different data types, to the Cell SPE's instruction set architecture (ISA).

One deficiency of the SPE is that it does not support scalar operations. Therefore, we did not add a full scalar ISA but rather a few scalar instructions that had proved to be most useful, such as load-scalar (into the preferred slot) and add-scalar-to-vector-element. Another deficiency of the Cell SPE is that it does not support clipping operations that saturate an operation's result. The SARC-MA supports clip, saturate-and-pack, and add-saturate-and-pack.

Often, a simple fixed-point operation is immediately followed by another simple fixed-point operation that depends on the first operation's result, causing a significant number of stall cycles. We can eliminate these stall cycles by collapsing these operations into a single operation, thereby allowing independent operations to continue in the pipe-line. The SARC-MA therefore supports several collapsed operations, such as add-and-shift, multiply-truncate, and multiply-add.

Another supported instruction is the swap-odd-even instructions, which swaps the odd-numbered elements of the first vector operand with the even-numbered elements of the second vector operand. This instruction accelerates matrix transposition.

Finally, the SARC-MA supports an intravector instruction that performs an 8-point 1D IDCT.

The instruction latencies have been estimated conservatively and have been based on the latencies of similar SPE instructions. Figure 2a shows the speedups and instruction count reductions achieved by the SARC-MA compared to the SPE for the considered H.264 kernels. These results show that we can obtain significant performance improvements by supporting a handful of application-specific instructions. These improvements directly translate to area cost savings. The full details are available elsewhere. [9]

Bioinformatics accelerator

The SARC bioinformatics accelerator (SARC-BA) is also an ASIP based on the Cell SPE. After inspecting the most time-consuming kernel of ClustalW and Smith-Waterman, we identified three new instructions that together significantly improve performance.

Computing the maximum between two or more operands is a fundamental operation often used in sequence-alignment kernels. Consequently, we added the Max instruction to the SPE's ISA to replace an operation that would otherwise need two SPE instructions. The analyzed kernel uses the Smith-Waterman recurrent formula that subtracts the penalties from the upper and left scores in the dynamic programming matrix. [10] Then, it computes the maximum value with saturation at zero. We also added two instructions to speed up this processing: Max3z computes the maximum of three input vectors and 0, and Submx computes max $\{a - b, c\}$.

Figure 2b depicts the speedups and instruction count reductions for the forward pass function, the most time consuming kernel of ClustalW. For Max and Max3z, the speedup is larger than the reduction in executed instructions. The reason is that these two instructions replace a sequence of instructions that create dependencies. Collapsing them into a single instruction saves many dependency stall cycles, further contributing to the total speedup. Overall, the new instructions improve performance by 17 percent.

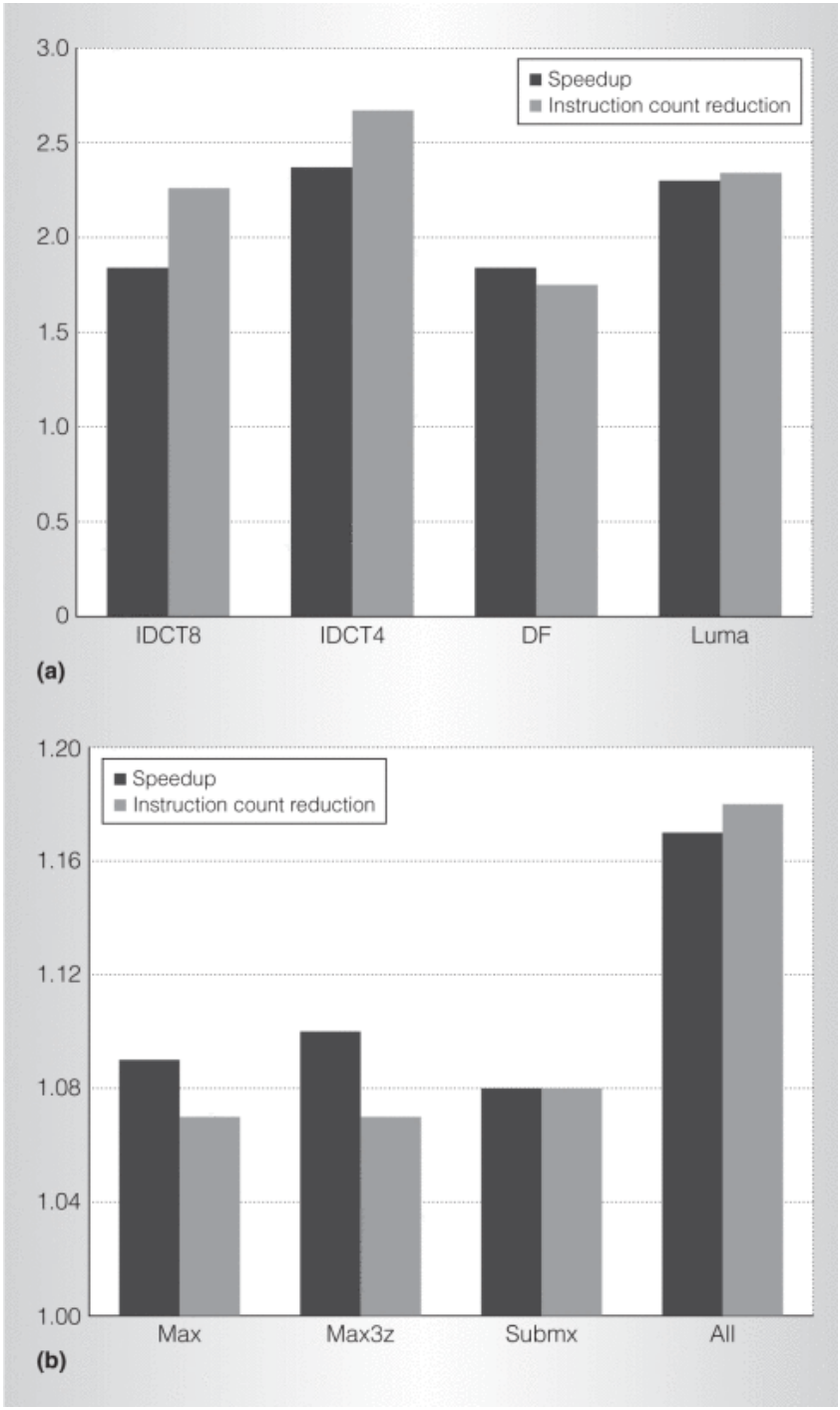

Figure 2. Speedup and dynamic instruction count reduction of the SARC media accelerator (SARC-MA) over the Cell synergistic processor element (SPE) (a) and of the SARC bioinformatics accelerator (SARC-BA) over the Cell SPE.

Scientific vector accelerator

The polymorphic register file plays a central role in the SARC scientific vector accelerator (SARC-SVA). Given a physical register file, the SARC-SVA lets us define 1D and 2D logical vector registers of different sizes and shapes. Figure 3 illustrates its organization, assuming that the physical register file contains 128 × 128 elements. When defining a logical vector

register, we need to specify its base address, horizontal length, and vertical length. The register file organization (RFOrg) special-purpose registers (SPRs) store the logical registers' parameters.

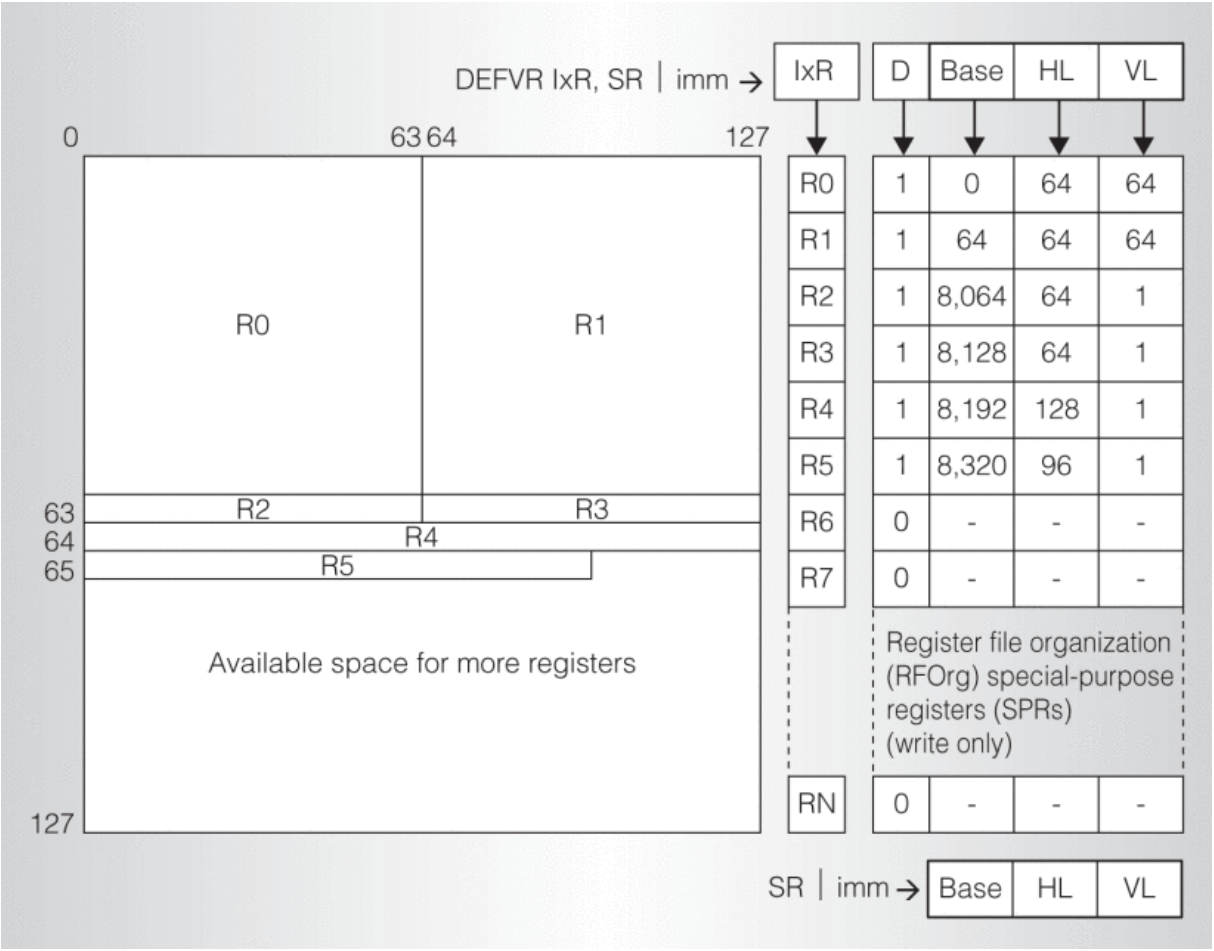

Figure 3. The polymorphic register file, that supports 1D-2D register operations.

The SARC-SVA micro architecture supports both 1D and 2D register operations simultaneously using the same instructions. Conditional execution is implicitly supported by defining a bit mask register for each logical register. By adding three extra bits to each RFOrg entry, we can also specify the data type (32/64-bit floating point or 8/16/32/64-bit integer) stored in the logic register, therefore avoiding the need to duplicate the instructions for each supported data type.

The Cell SPE implementation of the dense matrix multiplication is already highly optimized. IBM reports 98 percent efficiency for the hand-scheduled assembly implementation.[11] In addition, we need only 48 instructions compared to IBM code's 1,700. This will reduce the number of instructions dynamically executed by at least 35 times. Because this number does not change with the number of lanes, we are more interested in our code's efficiency. Figure 4 shows the performance results compared to an ideal (100 percent efficient) Cell SPE.

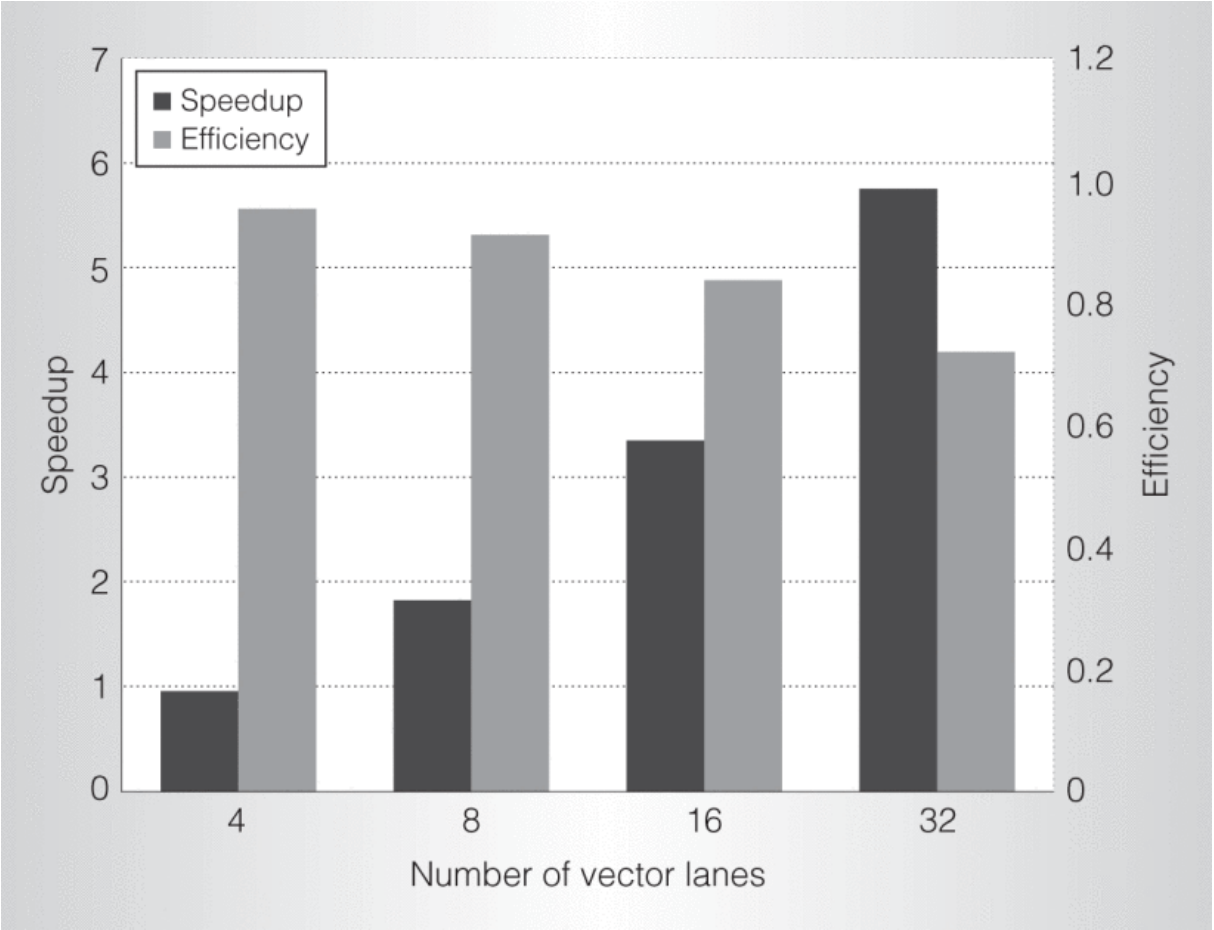

Figure 4. The SARC scientific vector accelerator (SARC-SVA) dense matrix multiplication performance and efficiency. Using 16 vector lanes, simulation results show a speedup of around 3 times. With 32 vector lanes, we estimate a speedup of approximately 5 times.

Figure 4 also shows that we can obtain similar efficiency at the synergistic processor unit (SPU) when using four vector lanes (95 percent). For 16 vector lanes, we estimate the efficiency at 84 percent. The reason for this behavior is that we did not use any blocking technique to overlap the local store loads and stores with computation. Therefore, the time required to perform memory transfers starts dominating the execution time when we use more vector lanes.

Performance Analysis

To evaluate the SARC architecture, we developed a trace-driven simulator called TaskSim. It provides cycle-accurate simulation of the entire SARC architecture, including the NoC, DMAs, caches, MICs, and DRAMs.

TaskSim is highly scalable because workers need not be simulated at the instruction level. Because task computation itself does not affect any component other than the worker (and

vice versa), tasks can be abstracted as atomic CPU bursts. TaskSim thus simply accounts for task computation time as if the worker executes a single instruction whose runtime is read from the trace file. TaskSim can therefore accurately model workers by only simulating their DMA transfers and task synchronizations, which are simulated alongside all other architectural components at the cycle level. This technique allows TaskSim to scale and accurately model the SARC architecture with hundreds of workers.

To guarantee that changing the number of simulated processors does not break application semantics, traces also include intertask dependency information. This information lets TaskSim group all tasks dispatched by the master in a single task list and dynamically schedule them to the simulated worker processors. The dependency information must verify the scheduling correctness, so that no task is scheduled before all its predecessors have finished, although the scheduling order might differ from that of the original trace because of the increased number of worker processors. Table 1 lists the baseline architectural settings used throughout this article.

The H.264 traces we used for this article correspond to the 2D-wave and 3D-wave processing of 100 frames of the pedestrian area video at full high-definition, 25 frames per second (fps) from HD-VideoBench[12] (see the "Target Applications" sidebar). The SARC H.264 instance uses one master processor and four CABAC processors. The number of processors in the scalability charts refers to the number of SARC-MA worker instances. The 2D-wave version can be seen as a specific case of the 3D-wave with only one frame in flight. The 3D-wave version supports a maximum of eight frames in flight.

We obtained the Smith-Waterman traces from a Cell implementation of the FASTA search algorithm. The SARC software instance uses one master processor and several SARC-BA worker instances.

We obtained the matrix multiply traces from a Cell implementation using the optimized 64 × 64 block multiplication kernel included in the Cell SDK. The SARC scientific instance uses one master processor, one helper processor, and several SARC-SVA worker instances.

Application	No. of tasks	Task duration (microseconds)	Bandwidth per task (GBps)	Problem size (Mbytes)
H.264	816,000	17.4	0.65	299
Smith-Waterman	3,670,016	50.3	0.65	20.7
Matrix multiply	262,144	25.8	1.42	192
Cholesky	357,760	28.0	1.68	512

Table 1. Baseline SARC simulation parameters.

Table 2 summarizes some important characteristics of the SARC target applications. We obtained the execution time of individual tasks in the corresponding SARC accelerators using a separate cycle-accurate CPU model that extends the Cell SPE with either the media or bio instructions and functional units, or implements the multidimensional vector scientific accelerator. To isolate the impact of worker specialization from the impact of parallel

scalability, all results presented correspond to the performance speedup relative to the SARC configurations with only one baseline Cell SPE worker.

Application	No. of tasks	Task duration (microseconds)	Bandwidth per task (GBps)	Problem size (Mbytes)
H.264	816,000	17.4	0.65	299
Smith-Waterman	3,670,016	50.3	0.65	20.7
Matrix multiply	262,144	25.8	1.42	192
Cholesky	357,760	28.0	1.68	512

Table 2. Characteristics of the SARC applications.

The use of local memories for workers in the SARC architecture isolates task execution, so tasks only interact with the worker's local memory and do not generate external memory traffic. None of the target SARC applications uses the Ll caches on the worker processors, because annotating the task inputs and outputs lets the runtime system automatically move all the data in and out of the local stores.

Parallel scalability

Figure 5 shows how performance improves as we increase the number of workers. The simulation parameters are those detailed in Table 1.

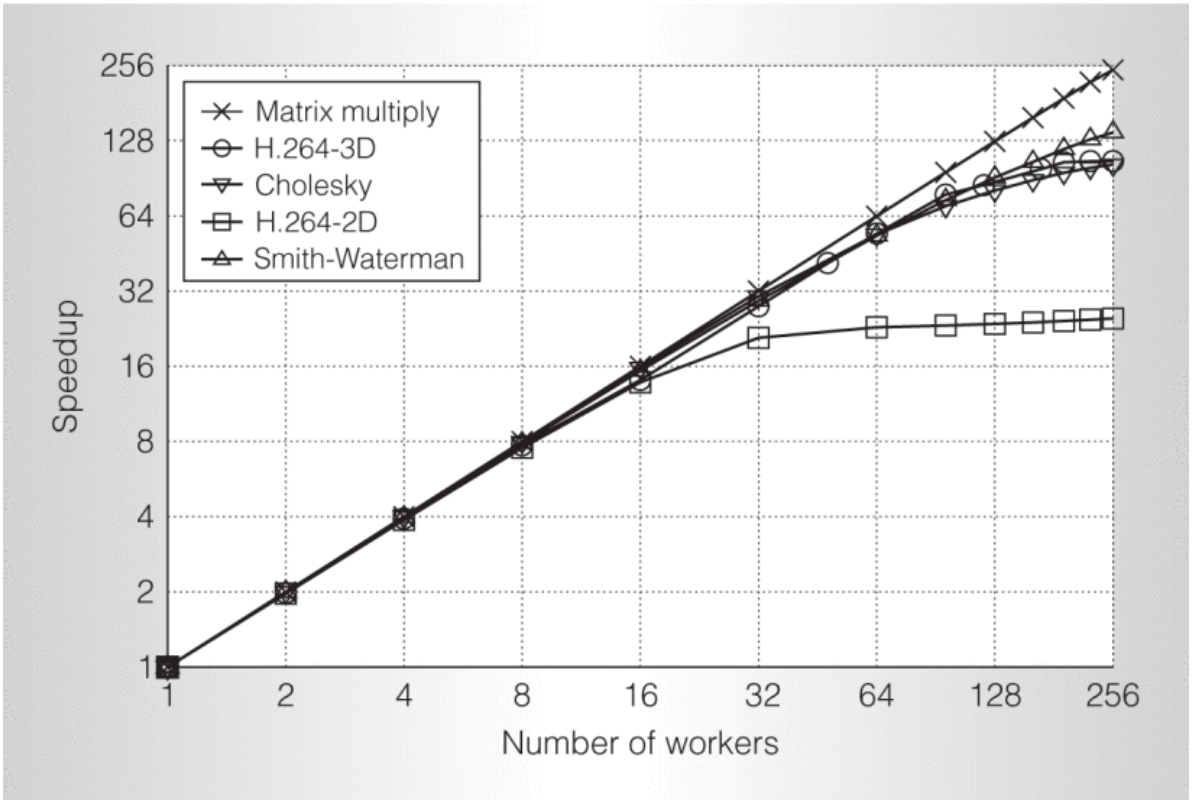

Figure 5. Parallel scalability of the SARC architecture. The graph shows the performance of the applications as the number of workers is increased.

Our results show perfect linear scalability for matrix multiply, our massively parallel benchmark. This shows that the application's parallelism has no architectural limitation. More importantly, it shows that the block multiply tasks do not incur significant memory stalls, as would be the case in any cache-based architecture.

The results for H.264, Cholesky, and Smith-Waterman show a different behavior. They scale linearly up to a point (16 workers for H.264–2D and 80 workers for H.264–3D, 96 for Cholesky and Smith-Waterman), and then only obtain diminishing returns. This limitations are not due to architecture bottle-necks; no resources are being overcommitted.

In Smith-Waterman, the sequential part starts to dominate the total duration. What was only 1 percent in the serial version accounts for 70 percent with 256 workers. In Cholesky, the number of parallel tasks diminishes as the algorithm progresses. H.264–2D simply lacks parallelism; the 2D-wave implementation has sufficient macroblocks for 60 workers in full High Definition (HD). The number of frames in flight (in this case, the value is limited to 8) limits the parallelism in H. 264–3D. Increasing the number of frames in flight will increase parallelism but result in a bigger memory footprint and higher memory bandwidth.

Impact of memory latency

Throughout this article, we have stressed that the SARC memory hierarchy is designed for high bandwidth, sometimes at the expense of latency. Such a design policy is based on the SARC runtime management system's ability to automatically schedule data transfers in parallel with the previous task's computation, achieving a double-buffering effect.

Figure 6a shows the performance degradation of our target applications as we increase the memory latency. (From this point on, all figures show the performance relative to the configuration with 256 workers.) For this experiment, we replaced the shared L2 and DDR memory system for an ideal conflict-free memory with a configurable latency ranging from 1 cycle to 16 K cycles. Note that the average DDR3 latency is between 150 and 250 cycles. On top of the raw memory latency is the added cost of traversing the SARC hierarchical NoC.

Our results show that performance for matrix multiply, Cholesky, and Smith-Waterman does not degrade until memory latency reaches 1 K cycles or higher. DMA transfers are 16 Kbytes in size, which requires 2,000 cycles at 8 bytes per cycle. An additional 1 K-cycle latency only increases total transfer time by 33 percent. Furthermore, double buffering helps to effectively hide latencies. Not only are higher latencies than the regular DDR3 DRAM tolerated, but the cache latency itself is completely irrelevant, because it will always be faster than the off-chip memory.

H.264’ s latency tolerance is much lower because it cannot fully benefit from the double-buffering runtime optimization. Not all the DMA transfers in the macroblock processing can be scheduled in advance, because the reference macroblock is not known until halfway through the decoding process. That is, the latency for accessing the reference block immediately translates to an increased macroblock decoding time. However, performance only degrades by 15 and 11 percent for a memory latency of 512 cycles for the 2D-wave and 3D-wave versions, respectively. The 3D-wave is more latency tolerant because transfers and computation from different frames can overlap.

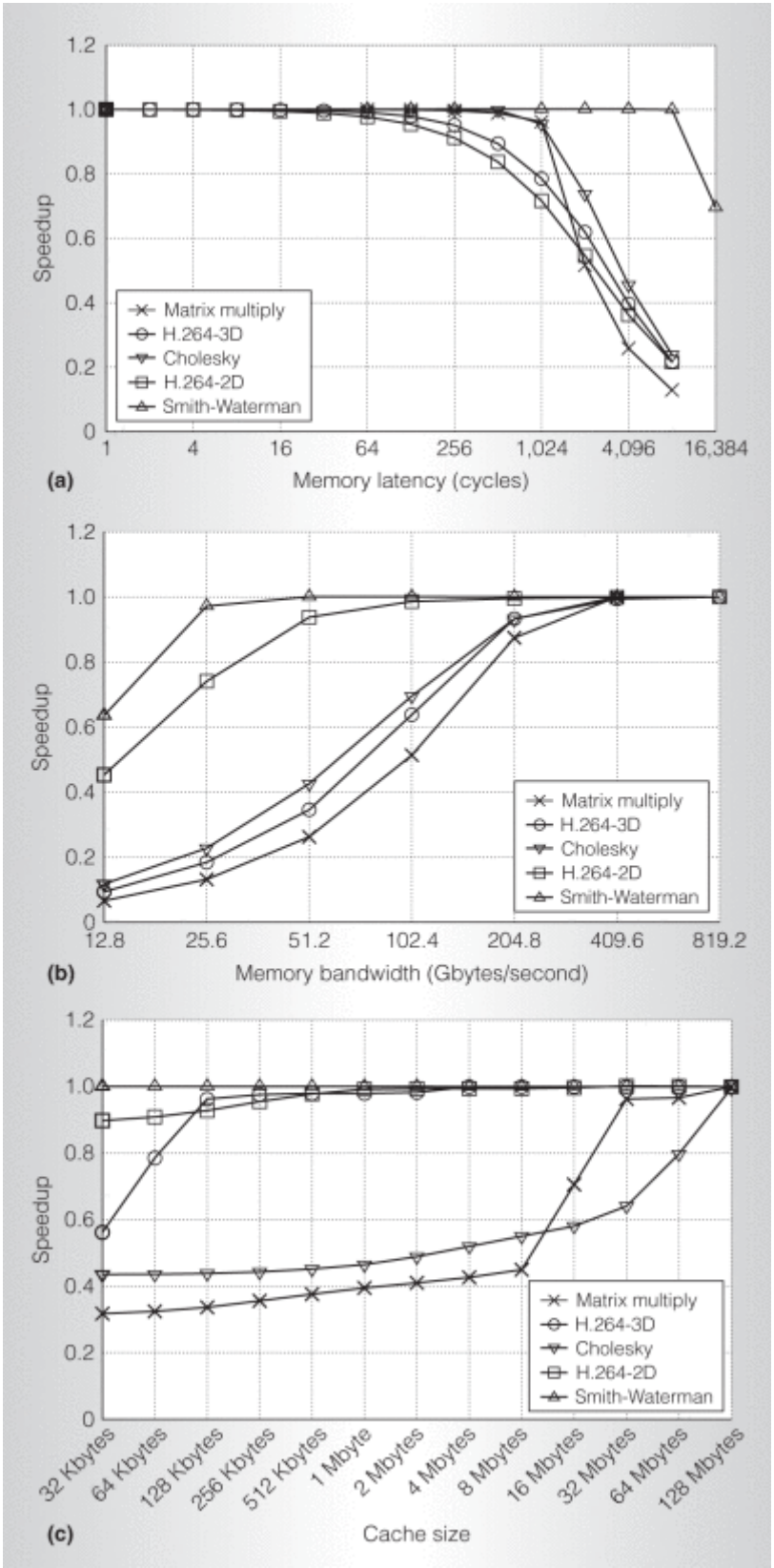

Figure 6. Impact of different architecture configuration parameter on the SARC architecture: memory latency (a), L2 cache bandwidth (b), and L2 cache size (c).

These results support our claims about the SARC architecture's latency tolerance. Such tolerance is not due to the architecture itself (even if bulk data transfers are dominated by bandwidth) but it would not be possible if the architecture does not let the DMA controllers overlap the transfers with computation.

Impact of memory bandwidth

Given our finding that performance is only limited by memory bandwidth, and not latency, we must provide sufficient bandwidth to the worker processors. For example, matrix multiply tasks take 25.8 microseconds, and require transferring up to 64 Kbytes of data in and out of the local memory (depending on data reuse). This translates to an average bandwidth requirement of 1.42 Gbytes per second per worker, and an estimated 363 GBps for all 256 workers.

Clearly, the limited pin count prevents us from providing such bandwidth from off-chip DRAM. We consider eight off-chi p DRAM channels to be a realistic design point, based on current pin restrictions. (The IBM Power6 already features two on-chip memory controllers and eight DRAM channels.) These would provide only 102.4 GBps using eight DDR3–1600 DIMMS. The distributed L2 cache must provide the rest of the bandwidth.

Figure 6b shows the performance degradation of the 256-worker SARC configuration as we reduce the L2 bandwidth from 32 L2 banks (819.2 GBps) to only one 12.8 GBps bank. Our results for matrix multiply, Cholesky, and H.264–3D show that they are the most bandwidth-demanding applications, as Table 2 suggests. Performance degrades about 10 percent for 204.8 GBps, and then drops dramatically for 102.4 GBps or lower bandwidth. H.264–2D and Smith-Waterman require much less memory bandwidth, and do not show significant performance degradation unless fewer than two L2 banks are used (51.2 GBps, half the off-chip bandwidth). H.264–3D scales more than H.264–2D at the cost of more memory bandwidth: going from 819.2 GBps to 102.4 GBps results in a 36 percent performance loss.

These results are a clear motivation for the fine-grained interleaving strategy we use in our multibank L2 cache. Although it leads to nonuniform latencies, it increases bandwidth, and our runtime system can hide the extra latency. As our results show, bandwidth is the critical factor limiting the SARC architecture's performance. We still need to check the cache's size requirements to capture the working set and avoid resorting to the limited off-chip pins.

Because we cannot provide sufficient off-chip bandwidth to support the large number of worker processors, most of the working set must be captured on-chip and serviced from the shared L2 cache. Distributing the cache across 32 blocks, each having multiple banks providing 25.6 GBps, gives sufficient concurrency to sustain the required 819.2 GBps bandwidth.

Impact of L2 cache size

Figure 6c shows how performance degrades as we reduce the L2 cache size from the baseline 128 Mbytes (32 blocks of 4 Mbytes each) to 32 Kbytes (32 blocks of 1 Kbyte each). The cache latency is fixed at 40 cycles, independent of the block size, but previous results have shown that performance does not depend on the latency. The off-chip bandwidth is limited to eight DRAM channels, or 102.4 GBps.

Our results show that Smith-Waterman's performance does not degrade as we reduce the cache size. This is to be expected, because previous results have shown that the off-chip 102.4 GBps are enough, even if all accesses miss in the cache.

Performance analysis for H.264–2D shows some performance degradation for cache sizes smaller than 1 Mbyte. The 1- Mbyte L2 size is sufficient to capture one reference frame, and so serves the latency-sensitive DMA transfer from cache instead of off-chip DRAM.

H.264–3D has a smaller performance degradation. This is due to the prefetch effect that results from processing multiple frames in flight. This effect appears because most of the motion vectors point to the colocated macroblock-that is, the macroblock in the reference frame that has the same coordinates as the current macroblock.

Finally, matrix multiply and Cholesky show a stronger dependency on cache size, as they are the most bandwidth-demanding applications. For matrix multiply, performance degrades significantly unless we can fit a whole row of matrix **A** and the full matrix **B** in the cache. This size depends on the workload and adds up to 64 Mbytes in our test case. Larger matrices would require more cache.

The Cholesky working set used for our experiments is too large to fit in cache (512 Mbytes), so we observe how larger caches can better capture the temporal locality of the irregular task dependency graph.

Our baseline configuration with 128 Mbytes only provides 512 Kbytes per worker. That is only twice the amount of local store available to them, and is a common size ratio in current multicore implementations. Given that 128 Mbytes would seem reasonable for a 256-worker chip, we conclude that the distributed on-chip L2 cache can effectively filter the off-chip memory bandwidth.

Impact of specialization

Figure 7a shows the SARC architecture's scalability, this time using the SARC domain-specific accelerators instead of the baseline worker processor.

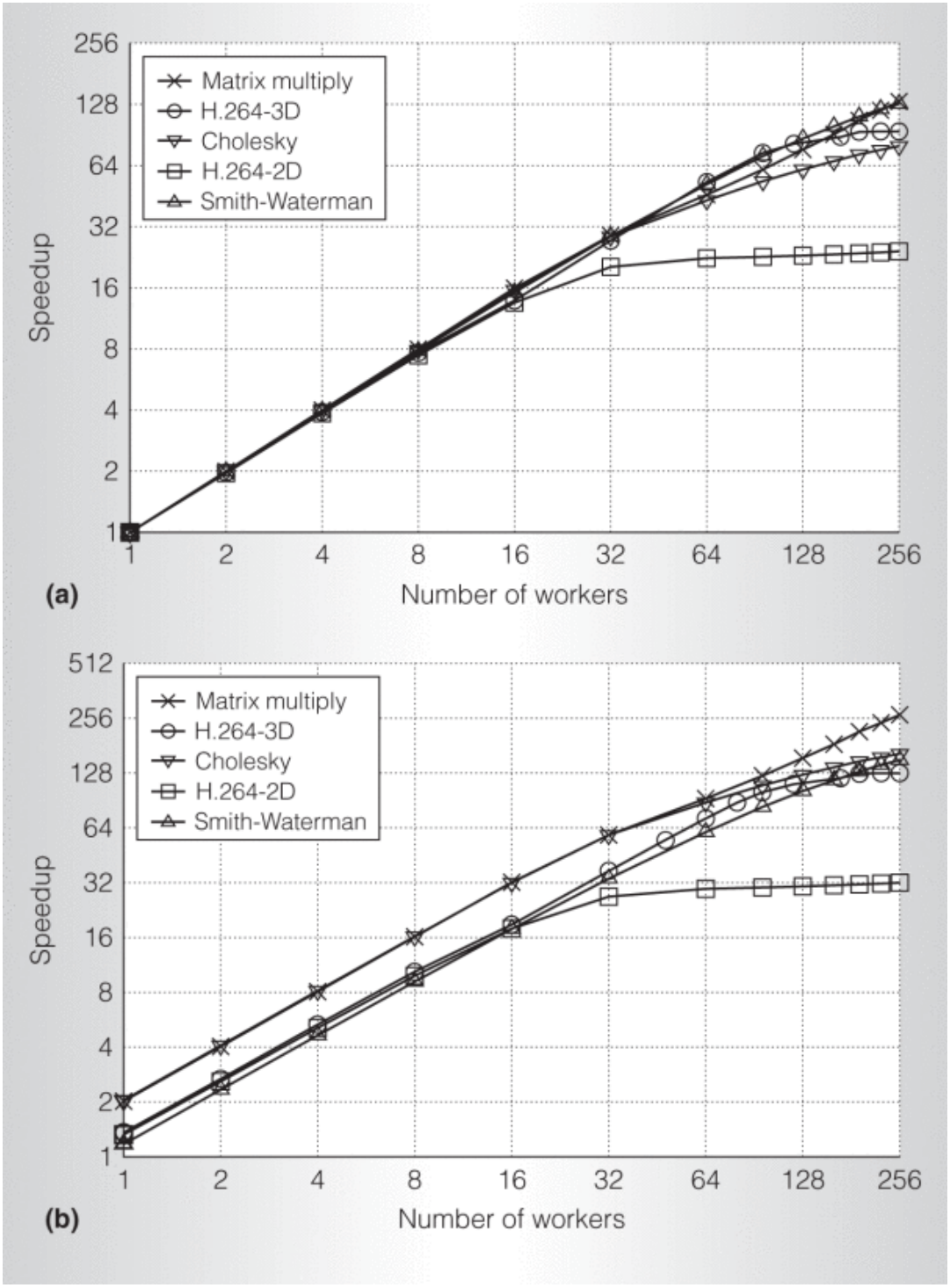

Figure 7. Scalability of the architecture with specialized accelerators: speedup measured against one accelerator (a) and speedup measured against one general worker processor (b).

Our results show worse parallel scalability than what was presented in Figure 5 for the baseline worker processor. Some applications scale to only half the number of processors as before. When using the faster SARC accelerators, the improved compute performance translates to increased bandwidth requirements from what is described in Table 2. Essentially, the 819.2 GBps of bandwidth provided by the L2 cache, and the 102.4 GBps of off-chip memory are not enough for such fast workers.

This is most visible for the most bandwidth-demanding benchmark, matrix multiply, for which the nonaccelerated architecture achieves almost perfect speedup for 256 workers (see Figure 8), whereas the accelerated architecture achieves only about 128.

We see the same effect for Cholesky, although to a lesser extent because it is less bandwidth demanding. Because the Smith-Waterman accelerator is only 17 percent faster than the base processor and because it is not very bandwidth demanding, the accelerated architecture provides a speedup of about 17 percent irrespective of the number of workers. H.264–2D has low bandwidth requirements, and also does not suffer any scalability impact due to the faster SARC-MA worker. For H.264–3D, the given bandwidth is enough to sustain the accelerated cores' scalability, but there is a 11 percent reduction in speedup compared to the nonaccelerated cores.

Figure 7b shows the combined impact of scalability and accelerators. This figure compares the performance speedup for the accelerated workers with the baseline Cell SPE worker. The bandwidth limitations encountered by faster workers prevent the SARC architecture from achieving speedups over 256 (for 256 workers). However, a comparison of the results to those in Figure 5 shows that SARC can achieve the same performance of the baseline architecture using fewer workers, translating to lower area and power requirements for a given performance target. H.264 also shows the impact of the improved worker, reaching a combined speedup of 32 and 128 times, for the 2D and 3D, that we could not achieve with the baseline workers.

The SARC architecture offers scalability with the number of workers and combines well with an heterogeneous set of domain-specific accelerators to achieve the desired performance level at a lower cost and power. We believe that the SARC architecture offers an excellent framework for continued research and development of scalable heterogeneous accelerator-based architectures and programming models.

Our experience developing this architecture shows that features such as data transfer engines (DMA in our case), heterogeneous processors (be it single ISA, or multi-ISA), distributed storage, and variable access latencies will be required for efficient designs. Equally important is that such features can be orchestrated by a smart runtime management layer, hiding the complexity to the programmer, and making this kind of design commercially viable.

Acknowledgment

We thank the rest of the team that developed the TaskSim simulator: Alejandro Rico, Carlos Villavieja, Augusto Vega, Toni Ques-ada, Milan Pavlovic, and Yoav Etsion. We also thank Pieter Bellens for his help obtaining the application traces. This research was supported by the SARC project (FP6-FET-27648), and the Consolider contract TIN2007–60625 from the Ministry of Science and Innovation of Spain. Felipe Cabarcas was also supported by the Program AlBan, the European Union Program of High Level Scholarships for Latin America (scholarship No. E05D058240CO). Finally, we recognize Manolis Katevenis for his participation in the definition of the SARC architecture, and Stamatis Vassiliadis and Mateo Valero, who made the SARC project happen in the first place.

References

1. R.M. Badia et al., “Impact of the Memory Hierarchy on Shared Memory Architectures in Multicore Programming Models,” Proc. 17th Euromicro Int’l Conf. Parallel, Distributed and Network-based Processing, IEEE CS Press, 2009, pp. 437-445.

2. R.D. Blumofe et al., “Cilk: An Efficient Multithreaded Runtime System,” Proc. 5th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP 95), ACM Press, 1995, pp. 207-216.

3. M.D. McCool et al., “Performance Evaluation of GPUs Using the RapidMind Development Platform,” Proc. Conf. Supercomputing (SC 06), ACM Press, 2006, p. 181.

4. K. Fatahalian et al., “Sequoia: Programming the Memory Hierarchy,” Proc. Conf. Supercomputing (SC 06), ACM Press, 2006, article 83.

5. A. Duran et al., “Extending the OpenMP Tasking Model to Allow Dependent Tasks,” Proc. Int’l Conf. OpenMP in a New Era of Parallelism (IWOMP 08), LNCS, Springer, 2008, pp. 111-122.

6. F. Quintana et al., “Adding a Vector Unit to a Superscalar Processor,” Proc. Int’l Conf. Supercomputing (SC 99), ACM Press, 1999, pp. 1-10.

7. M.G.H. Katevenis et al., “Explicit Communication and Synchronization in SARC,” IEEE Micro, vol. 30, no. 5, 2010, pp. xx-xx.

8. B. Jacob, S.W. Ng, and D.T. Wang, Memory Systems: Cache, DRAM, Disk, Morgan Kaufmann, 2008.

9. C.H. Meenderinck and B.H.H. Juurlink, “Specialization of the Cell SPE for Media Applications,” Proc. Int’l Conf. Application-Specific Systems, Architectures, and Processors, IEEE CS Press 2009, pp. 46-52.

10. T.F. Smith and M.S. Waterman, “Identification of Common Molecular Subsequences,” J. Molecular Biology, vol. 147, no. 1, Mar. 1981, pp. 195-197.

11. T. Chen et al., “Cell Broadband Engine Architecture and Its First Implementation: A Performance View,” IBM J. Research and Development, vol. 51, no. 5, 2007, pp. 559-572.

12. M. Alvarez et al., “HD-VideoBench. A Benchmark for Evaluating High Definition Digital Video Applications,” Proc. IEEE Workload Characterization Symp., IEEE CS Press, 2007, pp. 120-125.

About the Authors

Alex Ramirez

Barcelona Supercomputing Center

Alex Ramirez is an associate professor in the Computer Architecture Department at the Universitat Polirecnica de Catalunya (UPC), Barcelona, Spain, and leader of the Computer Architecture group at the Barcelona Super-computing Center. His research interests include heterogeneous multicore architectures and application-specific accelerators. Ramirez has a PhD in computer science from UPC.

Felipe Cabarcas

Barcelona Supercomputing Center

Felipe Cabarcas is a professor of electrical engineering at the Universidad de Antioquia, Medellin, Colombia, a PhD candidate in computer architecture at the Universitat Politecnica de Catalunya, and a research assistant at the Barcelona Supercomputing Center. His research interests include highperformance and multicore architectures. Cabarcas has a master's degree in electrical engineering from the University of Delaware.

Ben Juurlink

Technische Universitat Berlin

Ben Juurlink is a professor of embedded systems architectures in the Faculty of Electrical Engineering and Computer Science of the Technische Universitat Berlin. His research interests include multi-and manycore processors and memory systems. Juurlink has a PhD in computer science from Leiden University. He is a senior member of IEEE, a member of the ACM, and a member of the HiPEAC Network of Excellence.

Mauricio Álvarez-Mesa

Universitat Politecnica de Catalunya

Mauricio Alvarez is a PhD student in the Computer Architecture Department at the Universitat Politecnica de Catalunya, where he is a member of the High Performance Computing Group. His research interests include highperformance architectures for multimedia applications and streaming architectures. Alvarez has a BSc in electronic engineering from the University of Antioquia, Colombia.

Friman Sanchez

Universitat Politecnica de Catalunya

Arnaldo Azevedo is a doctoral candidate in the Computer Engineering Laboratory of the Faculty of Electrical Engineering, Mathematics and Computer Science of Delft University of Technology, the Netherlands. His research interests include multimedia accelerators architectures for multicore processors. Azevedo has an MSc in computer science from the Universidade Federal do Rio Grande do SuI, Brazil.

Arnaldo Azevedo

Delft University of Technology

Cor H. Meenderinck is a postdoctoral researcher in the Computer Engineering Laboratory of the Faculty of Electrical Engineering, Mathematics, and Computer Science at Delft University of Technology, the Netherlands. His research interests include multicore architecture and runtime systems. Meenderinck has a PhD in computer engineering from Delft University of Technology.

Cor Meenderinck

Delft University of Technology

Cătălin Ciobanu is a PhD student in the Computer Engineering Laboratory of the Faculty of Electrical Engineering, Mathematics and Computer Science of Delft University of Technology, the Netherlands. His research interests include computer architecture and parallel processing. Ciobanu has an MSc in computer engineering from Delft University of Technology.

Cătălin Ciobanu

Delft University of Technology

Sebastian Isaza is a PhD student in computer engineering at Delft University of Technology, the Netherlands, and is on the faculty at the University of Antioquia, Colombia. His research interests include multicore architectures and accelerators for bioinformatics workloads. Isaza has an MSc in embedded systems design from the University of Lugano, Switzerland.

Sebastian Isaza

Delft University of Technology

Friman Sanchez is a PhD student in the Computer Architecture Department at the Universitat Politecnica de Catalunya. His research interests include performance characterization of bioinformatics applications and parallel programming. Sanchez has a BSc in electronic engineering from the University of Antioquia, Colombia.

Gerogi Gaydadjiev

Delft University of Technology

Georgi Gaydadjiev is a faculty member in the Computer Engineering Laboratory, Microelectronics and Computer Engineering Department of Delft University of Technology, the Netherlands. His research interests include embedded systems design and advanced computer architectures. Gaydadjiev has a PhD in computer engineering from Delft University of Technology, the Netherlands. He is a member of the IEEE Computer Society and the ACM. Direct questions and comments about this article to Alex Ramirez, Campus Nord UPC, C6-E205, Jordi Girona, 1–3, 08034 Barcelona, Spain; alex.ramirez@bsc.es.