# Scalability Evaluation of a Polymorphic Register File: a CG Case Study

Cătălin B. Ciobanu[1], Xavier Martorell[2,3], Georgi K. Kuzmanov[1], Alex Ramirez[2,3], and Georgi N. Gaydadjiev[1]

[1] Computer Engineering Laboratory,
Electrical Engineering Department,
Delft University of Technology, The Netherlands
{c.b.ciobanu, g.k.kuzmanov, g.n.gaydadjiev}@tudelft.nl
[2] Universitat Politècnica de Catalunya, Spain
[3] Barcelona Supercomputing Center
{xavier.martorell, alex.ramirez}@bsc.es

**Abstract.** We evaluate the scalability of a Polymorphic Register File using the Conjugate Gradient method as a case study. We focus on a heterogeneous multi-processor architecture, taking into consideration critical parameters such as cache bandwidth and memory latency. We compare the performance of 256 Polymorphic Register File-augmented workers against a single Cell PowerPC Processor Unit (PPU). In such a scenario, simulation results suggest that for the Sparse Matrix Vector Multiplication kernel, absolute speedups of up to 200 times can be obtained. Moreover, when equal number of workers in the range 1-256 is employed, our design is between 1.7 and 4.2 times faster than a Cell PPU-based system. Furthermore, we study the memory latency and cache bandwidth impact on the sustainable speedups of the system considered. Our tests suggest that a 128 worker configuration requires the caches to deliver 1638.4 GB/sec in order to preserve 80% of its peak speedup.

## 1 Introduction

Recent generations of processor designs have reached a point where just increasing the clock frequency in order to gain performance is no longer feasible because of power and thermal constraints. As more transistors are available in each generation of CMOS technology, designers have followed two trends in order to improve performance: the specialization of the cores targeting improved performance in certain classes of applications and the use of Chip Multi-Processor (CMP) designs in order to extract more performance in multi-threaded applications. Examples of specialized extensions include Single Instruction Multiple Data (SIMD) extensions such as Altivec [9], which are designed to exploit the available Data Level Parallelism, but also the hardware support for the Advanced Encryption Standard [8] which provides improved performance for data encryption. A typical example of a heterogeneous CMP architecture is the Cell

processor [12]. This shift in the processor architectures employs new programming paradigms and has a significant impact on how programs have to be optimized in order to maximize performance. Engineers have to consider both single threaded performance but also multi-processor scalability. In our previous work we have proposed a Polymorphic Register File (PRF) [4], which provides an easier programming model targeting high performance vector processing.

More specifically, in this paper we investigate the scalability of such a PRF augmented vector accelerator when integrated in a multi-processor system. The study focuses on the achievable performance with respect to the number of processors when employed in a complex computational problem, namely the Conjugate Gradient (CG) method. CG is one of the most commonly used iterative methods for solving systems of linear equations[19]. The iterative nature of CG makes it a good option for solving sparse systems that are too large to be handled by direct methods. CG scalability is critical, as it determines the maximum problem size which can be processed within a reasonable execution time.

Previous studies have shown that 1D and 2D vector architectures can significantly accelerate the execution of this application - more than 10 times compared to a scalar processor [4]. In this work, we analyze the performance of such accelerators in a heterogeneous multicore processor with specialized workers - the SARC architecture [16]. Moreover, we consider critical parameters such as the available memory bandwidth and the memory latency. More specifically, the main contributions of this paper are the following:

- Performance evaluation of the Sparse Matrix Vector Multiplication (SMVM) kernel, comparing a vector processor using a Polymorphic Vector Register File implementation to the Cell BE and the PowerXCell 8i [10]. The Polymorphic vector register file system achieved speedups of up to 8 times compared to the Cell PowerPC Processor Unit (PPU);
- Scalability analysis of the SMVM kernel: simulation results suggest that a system comprising of 256 PRF accelerators can reach absolute speedups of up to 200 times compared to a single Cell PPU worker. The system scales almost linearly for up to 16 workers, and more than 50% of the single core relative speedup is preserved when using up to 128 PRF cores;
- Evaluation of the impact of memory latency and shared cache bandwidth on the sustainable performance of the SMVM kernel. We consider scenarios of up to 128 PRF workers and target at least 80% of their theoretical peak speedups. The memory latency simulations indicate that the system can tolerate latencies up to 64 cycles to sustain that performance. The cache tests suggest that such a configuration requires a bandwidth of 1638.4 GB/sec.

The rest of the paper is organized as follows: Section 2 provides the background information on the competitive architectures we have selected, the target application and describes related work. The case study scenario is presented in Section 3. Simulation data along with their analysis are presented in Section 4. Finally, the paper is concluded in Section 5.

## 2 Background and Related Work

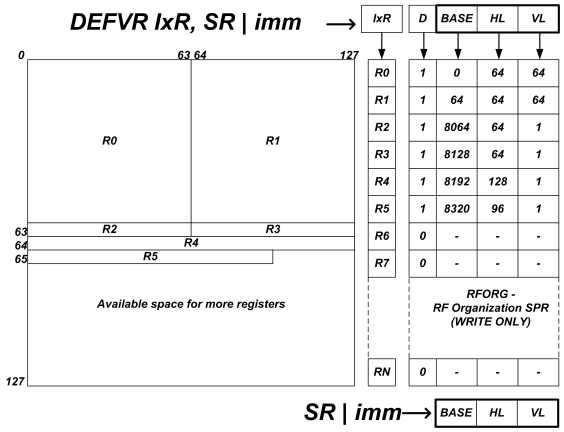



**Fig. 1.** The Polymorphic Register File

A **Polymorphic Register File** (PRF) is a parameterizable register file [4], which can be logically reorganized by the programmer or by the runtime system to support multiple register dimensions and sizes simultaneously. Figure 1 shows an example of a two-dimensional PRF assuming that the physical register file is 128 by 128 elements. The physical register storage space is allocated to a number of 1D and 2D logical vector registers, while remaining space is available for defining more logical registers. The benefits of this architecture are:

1. Potential performance gain by increasing the number of elements processed with a single instruction, due to multi-axis vectorization;
2. A more efficient utilization of the register file storage, eliminating the potential storage waste of fixed register size organizations;
3. Variable number of registers which can be defined in order to arbitrarily partition the available physical register space;
4. Reduced static code size as the target algorithm may be expressed with higher level instructions. The same binary instructions may be used regardless of the shape, dimensions or data type of the operands. The compatibility of the operands is checked by the microarchitecture.

The logical registers are defined by adding a second register bank to the architecture - the Register File Organization (RFORG) Special Purpose Registers (SPR). For each logical register, it is required to specify the coordinates: the location of the upper left corner (Base), the horizontal and vertical dimensions (Horizontal Length and Vertical Length) as well as the data type using a 3 bit

field, supporting 32/64-bit floating point or 8/16/32/64-bit integer values. More details on the organization of the PRF can be found in [4].

The **Conjugate Gradient Method** is one of the most important methods used for solving a system of linear equations, with the restriction that their matrix is symmetric and positive definite [19]. The iterative nature of the algorithm makes it suitable for solving very large sparse systems for which applying a direct method is not feasible.

The CG version we have used is part of the NAS Parallel Benchmarks [1]. By profiling the code we have found that the main computational kernel is the double precision Sparse Matrix - Dense Vector Multiplication (SMVM), which accounts for 87.32% of the total execution time in the scalar version of CG. The Compressed Sparse Row (CSR) format is used to store the sparse matrices. The following pseudo code sequence presents the SMVM kernel, where **a** is a one-dimensional array storing all the non-zero elements of the sparse matrix, **p** is the dense vector and **w** stores the result of the multiplication. **colidx** and **rowstr** contain the extra information required by the CSR format.

```
for (j = 1; j <= number_of_rows; j++) {
 w[j] = 0.0;
 lowk = rowstr[j]; upk=rowstr[j+1];
  for (k = lowk; k < upk; k++) {
   w[j] += a[k]*p[colidx[k]];
  }
}
```

By using the CSR format, the access to the vector **p** becomes irregular, making it difficult to vectorize the kernel unless a gather instruction is available in the architecture. Therefore, in order to vectorize the kernel, we first partitioned the code into an inspector and executor loop [6], effectively transforming the kernel into number of Dense Vector multiplications. The inspector loop gathers the data required to process a number consecutive rows of the sparse matrix from the global shared memory and places it into the Local Store of the worker, using DMAs.

In order to access a full row of the sparse matrix, a single DMA is sufficient as the non zeroes are stored in consecutive positions in vector **a**. However, in order to fetch the elements of the dense vector **p** on the positions stored into the column index array, multiple DMA requests are necessary as there is no predefined pattern of the positions of the non zero elements within a row of the matrix. Issuing a DMA for each element of the dense vector would be inefficient, requiring the issue of tens or hundreds of DMAs in order to multiply one row of the sparse matrix with the dense vector. Therefore, we have chosen to transfer the complete vector **p** to each worker before the actual computation of the SMVM. This way, the inspector loop always accesses the Local Store when accessing the dense vector. We have used a binomial tree broadcast algorithm, in which only the root node transfers the vector from main memory to its Local Store, and all the other transfers are Local Store to Local Store. In each step, all the

workers which already have a copy of the vector will participate in the transfers as sources. For P workers, $\log_2(P)$ steps are required to complete the broadcast.

After executor loop can be first vectorized in one dimension by performing each dot-product operation with one instruction. In order to vectorize the code on the second axis, multiple dense vectors are placed in a 2D vector register. Therefore, we have blocked the outer loop so that **block_size** rows of the matrix are processed in each step. The following pseudo code presents the reorganization of the kernel, with the **apriv** and **priv** buffers being allocated in the LS of the accelerator:

```
for (jj = 1; jj <= num_rows; jj+=block_size) {
//inspector loop
 for (j = jj; j < jj+block_size; j+=1) {
  lowk = rowstr[j], upk = rowstr[j+1];
  kp=0;
  for (k = lowk; k < upk; kp++, k++) {
   priv[j-jj][kp] = p[colidx[k]];
   apriv[j-jj][kp] = a[k];
  }
 }
//executor loop
 for (j = jj; j < jj+block_size; j+=1) {
  w[j] = 0.0;
  lowk = rowstr[j], upk = rowstr[j+1];
  for (k = lowk; k < upk; k++)
   w[j] += apriv[j-jj][k-lowk] * priv[j-jj][k-lowk];
 }
}
```

Previous work done in [4] assumed that all the necessary data to perform SMVM are available in the LS of the accelerator before the computation, a limitation which is not present in this study. However, we do assume that the size of the LS is sufficient to store a complete copy of the dense vector **p**.

**The SARC architecture** is a clustered heterogeneous multi-core architecture designed for the master-worker programming model. Figure 2 presents a block diagram of the architecture. A Network on Chip (NoC) connects multiple tiles (or clusters) of Workers (W), the Master (M) processor, the Memory Interface Controllers (MIC) and the shared L2 cache.

Each cluster may contain multiple Worker processors interconnected by a bus. In this work it is assumed that the main computational tasks are off-loaded to the worker processors. The workers have access to a private Local Store (LS) and use DMAs to transfer data from the global shared memory or from other worker's LS. The workers may be general purpose or specialized accelerators. By providing each worker with a private LS, there is no need for L1 caches for the worker processors.

The Master (M) processors are high performance, out-of-order superscalars which handle the dispatch of tasks to the workers and the control part of the

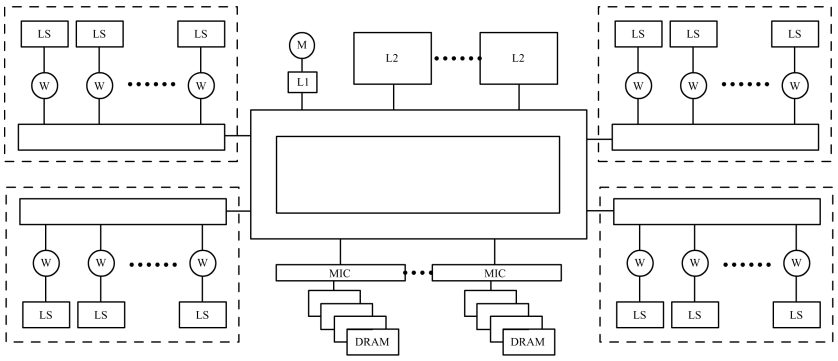



**Fig. 2.** The SARC architecture

program. Each Master processor is equipped with a private L1 cache which is coherent with the shared L2 cache. The architecture supports multiple Master processors, but since we offload the bulk of the computations to the worker clusters, we only instantiate one Master in this study.

Multiple Memory Interface Controllers provide access to the off-chip DRAMs, with each MIC supporting multiple DRAM channels. Fine-grain interleaving assures that when accessing consecutive memory addresses, the data are provided from multiple MICs and multiple DRAM channels in order to increase the effective bandwidth utilization. In order to exploit data locality, a shared multi-banked L2 cache is connected to the NoC, which also uses fine grain interleaving.

**Related Work**: The efficient processing of multidimensional arrays has been targeted by other architectures as well. One approach is to use a memory to memory architecture, such as the Burrows Scientific Processor (BSP) [13]. Being optimized for executing Fortran code, the ISA composed of high level vector instructions with a large number of parameters. The arithmetic units were equipped with 10 registers which are not directly accessible by the programmer. The Polymorphic register file also creates the premises for a high level ISA, but can reuse data directly within the register file. The Complex Streamed Instructions (CSI) [11] approach also did not make use of any data registers. CSI allows the processing of two-dimensional data streams of arbitrary length, but requires data caches to benefit from data locality. Our approach can use the register file in order to reduce the need for high speed data caches.

The Vector Register Windows (VRW) [14] concept allows the grouping of consecutive vector registers in a 2D window. However, one of the dimensions is fixed, contrary to our proposal. The Matrix Oriented Multimedia (MOM) [5] also uses a 2D register file, but with a fixed number of registers which used sub-word parallelism in order to store up to 16x8 elements. The Polymorphic register file also supports sub-word level parallelism but doesn't restrict the number or shape of the two dimensional registers. Modified MMX [18] supports

8 multimedia registers, each 96 bits wide. However, the matrix operations are constrained only to loads and stores.

The Register Pointer Architecture(RPA) [15] provides extra storage to a scalar processor by adding two additional register files - Dereferencible Register File (DRF) and the Register Pointers (RP). The DRF provides the extra storage space, while the RP provide indirect access to the DRF. The PRF also uses indirect accessing to a dedicated register file, but the RPA maps scalar registers, while in our proposal each indirection register maps to a matrix, being more suitable for vector processing.

In order to adjust the number of registers in a VLIW, FPGA partial reconfiguration is used to adjust the size of the physical register file in [20]. Our approach assumes that the physical register file is fixed, but offers a higher level view of the available storage space, eliminating many overhead instructions, possibly improving performance.

## 3 Case Study Scenario

We have used the Mercurium source-to-source compiler infrastructure [3] to obtain a parallel version of the CG benchmark. In such a parallel version, each of the parallel loops annotated with directives is transformed in a set of parallel tasks, each task executing a subset of the loop iterations. Examples of how the loops are annotated can be found in [7]. The resulting code contains calls to an instrumented runtime system that generates a trace with the relevant events, like task spawning, joining, and starting and ending of DMA transfers. While running the benchmark, the runtime system generates a file with the traced events, which is later analyzed to obtain the execution times of tasks, and the latencies added by the DMA transfers. Runs can be done in any serial system to obtain the traces.

Traces are later used as input to our simulation environment (Tasksim [17]), where the events are analyzed, and replayed, and the simulation environment changes timestamps according to the characteristics of the simulated architecture. Tasksim accurately simulates the timing of memory transfers, but the duration of the computational phases needs to be provided separately. Therefore, we have optimized the executor loop of the kernel for each accelerator type assuming that the required data are already present in the Local Store of the processor.

The normalized performance of the executor loop on a single core is presented in Figure 3, using the Cell PPU as the baseline processor. The results for the Cell and PowerXCell Synergistic Processor Units (SPUs) were obtained using the hardware counters in IBM QS21 and QS22 blades, compilation being performed using GCC 4.1.1 and IBM XLC 9.0 (optimization flags used are -O3 for GCC and -O5 for IBM XLC). In order to evaluate the PRF we used SARCSIM, a cycle accurate simulator [4], with a vector register size of 10x320 elements. It can be observed that the best results are obtained by using the IBM XLC compiler, which is between 68% and 259% faster than GCC. Therefore, throughout the paper, we shall only refer to the results obtained using the XLC compiler. The

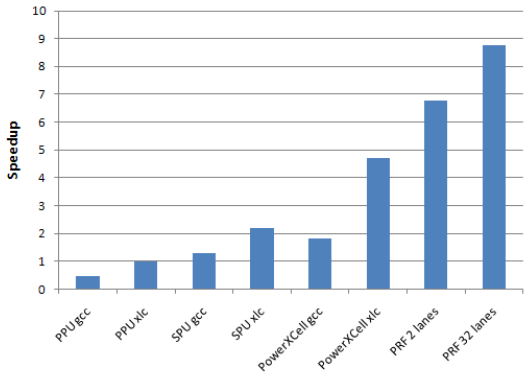



**Fig. 3.** CG - Normalized single core performance

Cell SPU is twice as fast compared to the Cell PPU, with the PowerXCell SPU providing a speedup of more than 4 times compared to the baseline, thanks to the improved double precision floating point arithmetic units. Using a PRF-augmented vector processor with 2 vector lanes or with 32 lanes accelerates the execution by more than 6 and respectively 8 times. The PRF implementation was based on the work presented in [4], implementing the updated kernel which uses the inspector/executor paradigm presented in Section 2.

**Table 1.** Tasksim simulation parameters

(a) Baseline configuration

| **Parameter** | **Value** |
|---|---|
| Clock Frequency | 3.2GHz |
| Memory Interface Contr. (MIC) | 16 |
| Memory channels | 2 DDR3 channels / MIC |
| Memory channel bandwidth | 12.8GB/sec (DDR3 1600) |
| Aggregate memory bandwidth | 409.6 GB/sec |
| Memory latency | Realistic DDR3 |
| L2 cache size | 128MB (128 banks) |
| L2 cache bandwidth | 25.6 GB/sec / bank |
| L2 cache associativity | 4 |
| L2 cache latency | 9 cycles |
| Local Store size | 256KB |
| Local Store latency | 6 cycles |

(b) L2 cache

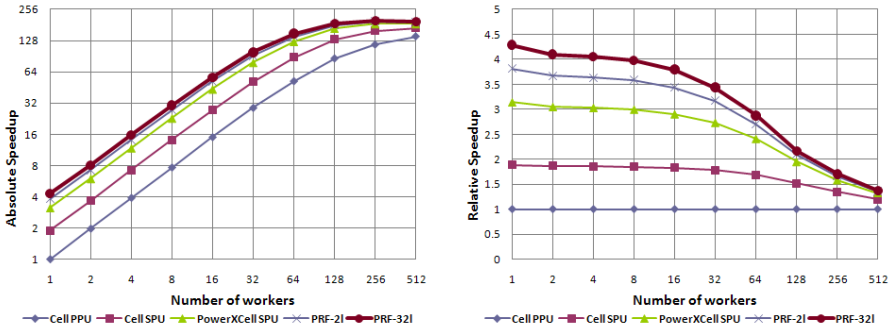
| **Bank size (KB)** | **Latency (cycles)** |
|---|---|
| 256 | 7 |
| 512 | 8 |
| 1024 | 9 |
| 2048 | 11 |
| 4096 | 13 |
| 8192 | 15 |
| 16384 | 20 |
| 32768 | 30 |
| 65536 | 50 |
| 131072 | 100 |

The multicore simulations use the default values presented in Table 1(a). We have focused on three scenarios with our experiments: the general scalability of

the kernel, the impact of the memory latency and shared L2 cache bandwidth on the sustainable speedups. The simulation results only include the Sparse Matrix Vector Multiplication kernel, which has been accelerated for the different worker types. We have used the Class A input data from the NAS Parallel Benchmark suite, in which the size of the sparse matrix is 14000 rows.

## 4 Simulation Results and Analysis

In this section, we investigate, through simulations, the scalability of the proposed system with respect to performance. Furthermore, we analyze the impact of the memory latency and the available L2 cache bandwidth on the overall performance of the PRF-enabled system by comparison to base-line configurations.




(a) Multicore scalability (b) The impact of specialized workers

**Fig. 4.** SMVM scalability

**Scalability** results of the SMVM kernel are presented in Figure 4, for five types of workers: Cell PPU, Cell SPU, PowerXCell SPU, PRF-augmented Vector Accelerator with 2 vector lanes (PRF-2l) and with 32 vector lanes (PRF-32l). Figure 4(a) presents the absolute speedups when the baseline performance is represented by a system with one Master and 1 Cell PPU worker processor, while Figure 4(b) shows the relative speedup obtained by each specialized worker type compared to a baseline system which uses an equal number of Cell PPUs. The baseline in Figure 4(b) corresponds to the lowest curve in Figure 4(a).

The system scales almost linearly for up to 16 workers, increasing the absolute speedup by more than 85% each time the number of processors doubles, regardless of the worker category considered. For less than 16 workers, the PRF systems maintain a relative speedup of around four, and the PowerXCell SPUs a relative speedup of around three.

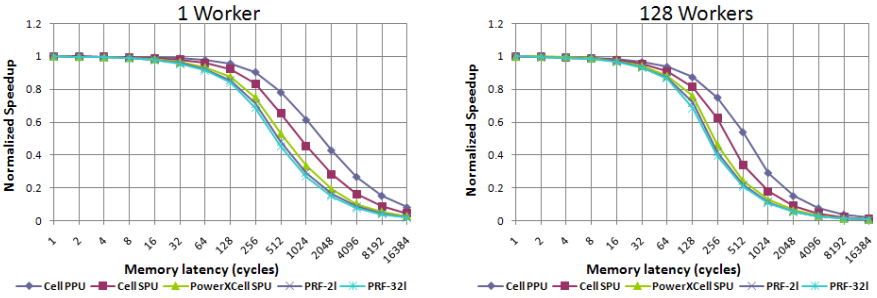
For 32 processors or more, the difference between the relative speedups of the worker types becomes narrower: the PRF accelerators maintain 50% of their

relative speedup advantage up to 128 workers, compared to 256 processors in the case of the Cell and PowerXCell SPUs. The absolute speedup saturates when using more than 256 Cell SPUs or more than 128 PRF cores.

The highest absolute speedups of the SMVM kernel are obtained when employing 256 PRF accelerators: the execution is accelerated by a factor of 200 with 32 lanes and 194 for 2 lanes. The peak absolute speedup of the PPUs is 140 for 512 workers, which is lower than the absolute speedup of 149 delivered by 64 PRF-32 lanes workers.

By analyzing the results with Paraver [2], we noticed that an important factor which prevents the system from achieving higher speedups when 512 workers were employed is the relatively small size of the input data. As the number of tasks available for each processor is reduced by increasing the worker count, it is increasingly difficult for the system to balance the workload between processors, because the number of non zero elements in each row of the sparse matrix is not constant. The small number of rows allocated to each worker also increases the relative cost of the broadcast of the dense vector. This operation duration is determined by the characteristics of the memory system, not by the worker type.

The simulation results suggest that using the PRF for this application is indeed scalable, as more than 50% of the single core relative speedup is preserved when using up to 128 cores, and the maximum absolute speedup obtained by 512 Cell PPU workers can be exceeded by 64 PRF-augmented Vector Accelerators, potentially saving area and power for a target performance level.




(a) (b)

**Fig. 5.** The impact of the memory latency on the performance of the SMVM kernel

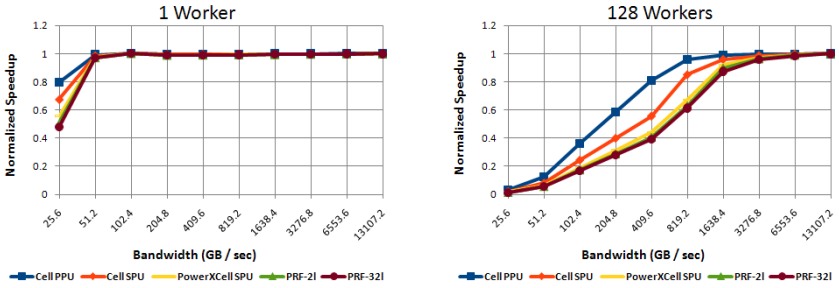
**Memory latency:** In order to evaluate the impact of memory latency on the performance of the SMVM kernel, an idealized, conflict-free, high bandwidth system featuring 256 MICs with 2 DRAM channels each, with a total memory bandwidth of 6553.6 GB/sec is simulated. The L2 cache is disabled and the memory latency is artificially fixed to a value between 1 to 16384 cycles. The speedups for each worker type are normalized to the theoretical peak speedup

delivered by that processor type, which is determined by assuming a perfect memory system with 1 cycle latency.

In the single worker scenario (Figure 5(a)), all the worker types can sustain more than 90% of the theoretical peak speedups for a memory latency smaller than or equal to 64 cycles. While the PPU can sustain 80% of its theoretical peak performance for a memory latency of 512 cycles, the SPU can only achieve this for a latency of 256 cycles, and 128 cycles in the case of the PRF. When fixing the latency to 512 cycles, the normalized speedups drop to 78% of their theoretical peak for the Cell PPU, 65%-for the Cell SPU, and to 45%-for the processor using the 32-lanes PRF.

When 128 workers are employed (Figure 5(b)), for a fixed memory latency of 32 cycles or less, the normalized speedups stay above 90% regardless of the type of cores. A minimum of 80% of its theoretical peak speedups can be sustained for a memory latency of up to 128 cycles for the slowest worker type, the Cell PPU and up to 64 cycles for the PRF. A 512 cycles memory latency reduces the normalized speedup to 54% in the PPU case and 20% in the case of the PRF.

The simulations suggest that using multicore configurations and/or faster workers require lower memory latency in order to sustain a fixed proportion of their own theoretical peek speedup.




(a) (b)

**Fig. 6.** The impact of the L2 cache bandwidth on the performance of the SMVM kernel

**L2 cache bandwidth:** In order to evaluate the influence of the available L2 cache bandwidth, the total size of the L2 cache is fixed to 128MB, and the number of cache banks is varied from 1 to 512. The latencies used for the cache according to the size of the banks are presented in Table 1(b). As in the scalability test, realistic memory latencies are assumed. The speedups reported are normalized to the theoretical peak speedup of each worker type, achieved in this case for a very high L2 cache bandwidth of 13107.2 GB/s for 512 banks.

When a single worker is employed, a cache bandwidth of 51.2 GB/sec is sufficient to sustain 99% of the PPU's and 97% of the PRF's peak speedups, as

suggested by Figure 6(a). For bandwidths in excess of 51.2 GB/sec, the speedups saturate regardless how fast the worker processor is.

Figure 6(b) suggests that for 128 workers to sustain 80% of their peak speedup, 16 cache banks providing 409.6 GB/sec are required for the PPUs, and 64 cache banks for the PRFs. In order to perform within 5% of their peak speedup, the PPUs require the use of 32 cache banks supplying 819.2 GB/sec, while 128 cache banks are need by the PRFs to deliver 95% of their peak speedup.

## 5 Conclusions

We evaluated the scalability of a system facilitated by a Polymorphic Register File, using the CG method as a case study. Furthermore, we analyzed the impact of the memory latency and cache bandwidth on the sustainable speedups of the system considered. Simulation results suggested that compared to a single Cell PPU, significant speedups could be achieved for the SMVM kernel when integrating the PRF in a competitive heterogeneous multicore architecture employing up to 256 cores. Moreover, a target performance level could be achieved using fewer PRF cores compared to a Cell PPU-based system, potentially saving area and power. This proves that the PRF is suitable not only for small scale systems with few processing elements, but it can be successfully embedded in high performance many-core implementations, as well. In the future, we plan to analyze the PRF scalability in a wider range of applications, as well as evaluate the feasible microarchitectural implementations in terms of performance, power and hardware complexity.

## Acknowledgements

This work was partially supported by the European Commission in the context of the Scalable computer ARChitectures (SARC) integrated project #27648 (FP6), the HiPEAC-2 Network of Excellence (FP7/ICT 217068), the Dutch Technology Foundation STW, applied science division of NWO, the Technology Program of the Dutch Ministry of Economic Affairs (project DCS.7533), the Consolider contract TIN2007-60625 from the Ministry of Science and Innovation of Spain and the Generalitat de Catalunya (project 2009-SGR-980).

## References

1. D. Bailey, J. Barton, T. Lasinski, , H. Simon, and eds. The NAS Parallel Benchmarks. Technical Report Technical Report RNR-91-02, NASA Ames Research Center, Moffett Field, CA 94035, 1991.
2. Barcelona Supercomputing Center. Paraver. http://www.bsc.es/paraver.
3. Barcelona Supercomputing Center. The NANOS Group Site: The Mercurium Compiler. http://nanos.ac.upc.edu/mcxx.

4. C. Ciobanu, G. K. Kuzmanov, A. Ramirez, and G. N. Gaydadjiev. A Polymorphic Register File for Matrix Operations. In *Proceedings of the 2010 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS 2010)*, pages 241–249, July 2010.
5. J. Corbal, R. Espasa, and M. Valero. MOM: a Matrix SIMD Instruction Set Architecture for Multimedia Applications. In *Proceedings of the ACM/IEEE SC99 Conference*, pages 1–12, 1999.
6. R. Das, M. Uysal, J. Saltz, and Y. shin Hwang. Communication Optimizations for Irregular Scientific Computations on Distributed Memory Architectures. *Journal of Parallel and Distributed Computing*, 22:462–479, 1993.
7. R. Ferrer, M. González, F. Silla, X. Martorell, and E. Ayguadé. Evaluation of memory performance on the Cell BE with the SARC programming model. In *MEDEA '08: Proceedings of the 9th workshop on MEmory performance*, pages 77–84, New York, NY, USA, 2008. ACM.
8. S. Gueron. Intel Advanced Encryption Standard (AES) Instructions Set. http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-aes-instructions-set/, 2010. Available online.
9. L. Gwennap. AltiVec Vectorizes PowerPC. *Microprocessor Report*, 12(6):1–5, May 1998.
10. IBM. *Cell Broadband Engine Programming Handbook Including the PowerXCell 8i Processor*, 1.11 edition, May 2008.
11. B. Juurlink, D. Cheresiz, S. Vassiliadis, and H. A. G. Wijshoff. Implementation and Evaluation of the Complex Streamed Instruction Set. *Int. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pages 73 – 82, 2001.
12. J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the Cell Multiprocessor. *IBM J. Res. Dev.*, 49(4/5):589–604, 2005.
13. D. Kuck and R. Stokes. The Burroughs Scientific Processor (BSP). *IEEE Transactions on Computers*, C-31(5):363–376, May 1982.
14. D. Panda and K. Hwang. Reconfigurable Vector Register Windows for Fast Matrix Computation on the Orthogonal Multiprocessor. In *Application Specific Array Processors, 1990. Proceedings of the International Conference on*, pages 202 –213, 5-7 1990.
15. J. Park, S.-B. Park, J. D. Balfour, D. Black-Schaffer, C. Kozyrakis, and W. J. Dally. Register Pointer Architecture for Efficient Embedded Processors. In *DATE '07: Proceedings of the conference on Design, Automation and Test in Europe*, pages 600–605, San Jose, CA, USA, 2007. EDA Consortium.
16. A. Ramirez, F. Cabarcas, B. Juurlink, M. Alvarez Mesa, A. Azevedo, C. Meenderinck, G. Gaydadjiev, C. Ciobanu, S. Isaza, and F. Sanchez. The SARC Architecture. *Micro, IEEE*, 30(Issue:5):16 –29, Sept.-Oct. 2010.
17. A. Rico, F. Cabarcas, A. Quesada, M. Pavlovic, A. Vega, C. Villavieja, Y. Etsion, and A. Ramirez. Scalable Simulation of Decoupled Accelerator Architectures. Technical report, Universitat Politècnica de Catalunya, Barcelona, Spain, 2010.
18. A. Shahbahrami, B. Juurlink, and S. Vassiliadis. Matrix Register File and Extended Subwords: Two Techniques for Embedded Media Processors. In *Proceedings of the 2nd ACM Int. Conf. on Computing Frontiers*, pages 171–180, May 2005.
19. J. R. Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.
20. S. Wong, F. Anjam, and M. Nadeem. Dynamically Reconfigurable Register File for a Softcore VLIW Processor. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE 2010)*, March 2010.