Efficient Execution of SkePU Skeleton Programs on the Low-power Multicore Processor Myriad2

Sebastian Thorarensen*, Rosandra Cuello*, Christoph Kessler*, Lu Li* and Brendan Barry†
*IDA, Linköping University, Linköping, Sweden
Email: {sebth181@student.|roscu097@student.|christoph.kessler@|lu.li@}liu.se
†Movidius Ltd., Dublin, Ireland
Email: brendan.barry@movidius.com

***Abstract*—SkePU is a state-of-the-art skeleton programming library for high-level portable programming and efficient execution on heterogeneous parallel computer systems, with a publically available implementation for general-purpose multicore CPU and multi-GPU systems. This paper presents the design, implementation and evaluation of a new back-end of the SkePU skeleton programming library for the new low-power multicore processor Myriad2 by Movidius Ltd. This enables seamless code portability of SkePU applications across both HPC and embedded (Myriad2) parallel computing systems, with decent performance, on these architecturally very diverse types of execution platforms.**

***Keywords*-SkePU, Myriad2, Skeleton Programming, Energy Efficiency, Embedded Chip, Multicore computing, Heterogeneous computing**

I. Introduction

Today parallel computing has become ubiquitous in our desktop machines, supercomputers and embedded systems. However, programming of parallel machines is notoriously challenging, as programmers need to explicitly handle problem decomposition, communication, synchronization etc. Moreover, mainly due to the power usage and heat dissipation issue, there is a trend towards more heterogeneous and architecturally diverse designs of processors and computer systems, which are exposed to programmers in the form of diverse and complex programming models, often at a relatively low level of abstraction. This enforces compromises between code portability, program performance and programmer productivity.

Embedded multicore chips such as Movidius Myriad2 are designed with strict low-power constraints, and can even run many scientific computations at a remarkable energy efficiency. For instance, Myriad2 [1], [2] by Movidius Ltd. is an embedded processor developed for mobile vision applications that achieved an energy efficiency of 50 GFlops/W, which is by one to two orders of magnitude better if comparing to general purpose processors such as current server CPUs and high-end GPGPUs used in HPC today. On the quest for eventually achieving exascale performance within an affordable power envelope, it thus becomes an interesting option to offload HPC computing work to (lots of) embedded chips.

Skeleton programming [3], [4] is an important and popular way to improve programmer productivity and portability by raising the level of abstraction in programming parallel computer systems, even addressing heterogeneous systems (such as systems with GPGPUs). Skeletons are pre-defined generic program building blocks based on higher-order functions such as map (data parallel computations), reduce (reductions), scan (prefix computations), stencil computations etc. that implement certain frequently occuring patterns of computation, that can be parameterized in problem-specific sequential code, and that have expert-written parallel implementations for the target architecture. Skeletons hide all aspects of programming for parallelism, heterogeneity, communication and synchronization from the programmer and handle them internally. This offers a solution to the programmability and portability issue, at least for computations that can be expressed in terms of the available skeletons. Today many commercial programming environments for multicore, GPUs, and distributed systems (e.g. Intel TBB, Nvidia Thrust, Google MapReduce) provide some form of skeleton constructs to the programmer.

SkePU [5] is a state-of-the-art skeleton programming framework for heterogeneous systems. It hides all parallelism, heterogeneity, communication and synchronization details from the programmer by providing back-ends (implementation variants) of pre-defined generic computations (skeletons) and performs automatic optimizations for performance and energy efficiency, such as automatically tuned context-dependent backend selection and lazy memory copying by smart containers) on runtime and energy automatically.

In this paper (more details of this work can be found in [6]) we demonstrate that the skeleton approach provides universal portability of skeleton programs across three classes of architecturally very different computer architectures, namely general-purpose multicore CPUs, general-purpose GPUs, and the low-power embedded multicore processor Myriad2. SkePU supports multicore CPU and multi-GPU systems with low abstraction overhead already since its first version [5]. In this work, we developed a new SkePU back-end for execution on Myriad2. We motivate its design, explain how technical challenges were solved in the implementation, and show that it can run SkePU skeleton code with decent performance on Myriad2. To the best of our knowledge, SkePU is thus the first skeleton framework to support portability and efficient execution across all three types of execution platforms.

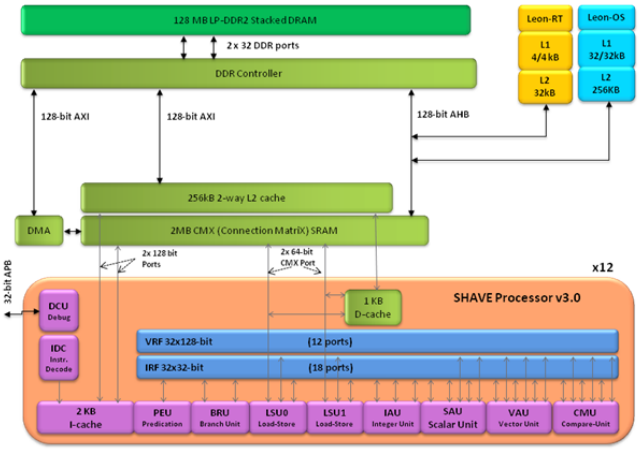

Fig. 1. Myriad2 Architecture

II. SkePU

SkePU[1] [5] is a C++ based skeleton programming framework with special support for efficient execution on heterogeneous parallel systems, most notably GPU-based systems. SkePU provides currently 7 general-purpose data-parallel skeletons (Map, Reduce, MapReduce, MapArray, MapOverlap (stencil), Scan and Generate). All skeletons have multiple implementation variants (back-ends) for different processor types and programming platforms, including sequential C++ and OpenMP for multicore CPUs and OpenCL and CUDA for single- and multi-GPU execution. With its *smart containers* support [7] the vector and matrix containers transparently implement a coherent software caching mechanism, monitoring the location and validity of elements at arbitrary sub-array granularity, which can greatly reduce unnecessary data transfers between memory of processors of different types, reduce memory allocation overheads, and automatically leverage direct GPU to GPU communication where available. The whole programming interface is (looks) sequential, and all platform-specific details (problem decomposition, heterogeneity, memory management, communication, synchronization etc.) are hidden in container and skeleton code.

III. Myriad2 Processor

The Myriad 2 processor [1], [2] (Figure 1) from Movidius is a multicore always-on System-on-Chip supporting computational imaging and visual awareness for mobile, wearable, and embedded applications. Also called a Vision Processing Unit (VPU), Myriad 2 is based on the twelve 128-bit SHAVE vector-processors, 2 Leon4 RISC processors, a hardware acceleration pipeline backed by a shared multiport memory subsystem and peripherals, and is fabricated on 28nm HPC process technology.

In order to guarantee sustained high performance and minimise power, the SHAVE processor contains wide and deep register-files coupled with a Variable-Length Long Instruction-Word (VLLIW) controlling multiple functional units including extensive SIMD capability for high parallelism and throughput at both a functional unit and processor level. The SHAVE processor is a hybrid stream processor architecture combining the best features of GPUs, DSPs and RISC with both 8/16/32 bit integer and 16/32 bit floating point arithmetic as well as unique features such as hardware support for sparse data structures. The architecture is designed to maximise performance/watt while maintaining ease of programmability, especially in terms of support for design and porting of multicore software applications.

IV. Design of the SkePU Back-end for Myriad2

A first attempt to designing a back-end for SkePU was done for the predecessor chip Myriad1, see Cuello [8]. The main approach investigated there was to run the main SkePU application code on the host computer (a standard PC) and execute only the SkePU skeleton invocations on Myriad1. An advantage of this design is that, in principle, the back-end could still be selected for each skeleton invocation among Myriad1 and execution on the host, e.g. on CPU or on GPUs, giving much flexibility to selection tuning; also hybrid execution involving both Myriad1 and host resources would, in principle, be possible. Moreover, running the whole C++ based SkePU application on the Leon master core was not possible at that time as there was no C++ execution support available when fixing the design [8]. However, this setup requires, at each skeleton call invocation, uploading the kernel code from host memory into Myriad device memory, before executing it, through a (comparably slow) host interface, which added tremendous overhead that was by far dominating the execution time of smaller SkePU programs [8]. The external loading overhead was the main reason for abandoning this design when targeting Myriad2 in this work.

A. Reduction of Program Loading Overhead

With the new back-end design, execution of a SkePU program starts on Myriad2's main (*LEON-OS*) master core (from DDR memory); skeleton computations are offloaded onto Myriad2's SHAVE worker cores using CMX memory, with dynamic loading. If the user chooses the CPU-backend explicitly, skeletons will run sequentially on LEON-OS. By not using an external host computer's CPU for controlling SkePU program execution, we remove the overhead of data transfer between host memory and the embedded chip which is a severe performance bottleneck. A potential disadvantage of this design is that it eliminates the option to use host CPU cores or host GPUs as alternative execution resources (or even hybrid execution) for some skeleton calls within the same application execution.

B. Runtime Loading of Skeletons

The linker used for linking Myriad2 applications did, up to now, not support the sharing of symbols between code for the LEON core and code for the SHAVE cores, when using

[1]For SkePU download (open source) and documentation, see http://www.ida.liu.se/labs/pelab/skepu

dynamically loadable SHAVE applications.[2] As a solution, addresses to memory areas shared by LEON core and SHAVE cores are hard-coded in the SkePU library code, instead of relying on the linker to assign the addresses. This is possible because both LEON and SHAVE applications are allowed to use absolute hardware addresses for accessing memory, and the hardware addresses are well defined in Myriad2.

It was found to be hard to signal data type and user function (which also is a type, namely, a struct [5]) from LEON to SHAVE as code for the LEON core and code for the SHAVE cores are compiled separately. All data types and user functions are assigned a unique id (an integer), which the LEON core sends to each SHAVE core when a skeleton is invoked. However, in the SHAVE code, a function template is used to generate the skeleton code for different data types and user functions. The type ids are received as variables, and cannot directly be used as template parameters, as template parameters must be constant. As a solution, template metaprogramming is used to generate conditional calls to the function template, with different parameters, for all possible values and combinations of the type ids.

C. Utilization of SIMD

The SHAVE processors are capable of 128-bit vector operations, by the use of SIMD. Movidius' compiler is capable of automatically generating SIMD instructions for code that is operating on vectors, without the need for any special C extensions or inline assembler [9]. By inlining[3] user functions into skeletons, SkePU-generated code allows the compiler to perform auto-vectorization on skeletons where it is applicable, if the user function contains operations corresponding to available SIMD instructions.

D. Overlapping Data Transfer and Computations

Most data transfers between the DDR and CMX memory during the execution of a skeleton are using DMA. Currently, all skeletons except for unary Map, binary Map, and MapOverlap2D for matrices, use synchronous DMA; the SHAVE cores do not perform any calculations while operands or results are being transferred. In unary Map, binary Map, and MapOverlap2D for matrices, background DMA transfers [4] are done (using multiple buffers) while calculations are being performed, which increases performance. transferred. In unary and binary Map, background multiple buffers) while calculations are being performance.

E. Smart Containers

Myriad2 contains memory caches for the LEON and SHAVE cores. Each LEON core has its own cache, and the SHAVE cores have one shared cache. When sharing data between the different cores, these caches must be kept coherent by the programmer. In the Myriad2 back-end, this is solved by using the smart containers feature of SkePU. The SHAVE cache is flushed after each skeleton invocation, but the LEON cache is refreshed on-demand. If a small vector that has been modified by the SHAVE cores is accessed by the LEON core, the vector is refreshed by iterating over the vector and refreshing each element with a special instruction that bypasses the cache. If a modified large vector is accessed, the entire data cache is flushed as it would be wasteful to iterate over a large vector.

Some skeletons, for instance MapArray, use a portion in the CMX memory, which is shared by all SHAVE cores, to store either a vector or a matrix to be accessible by the skeleton's user function during the entire execution of the skeleton. At the beginning of skeleton invocation, the vector or matrix to be shared is copied from the DDR memory to the shared portion of the CMX memory. To avoid redundant transfers to the shared memory during subsequent skeleton invocations, smart containers are used to keep track of the state of the vector or matrix currently having a copy in the shared memory.

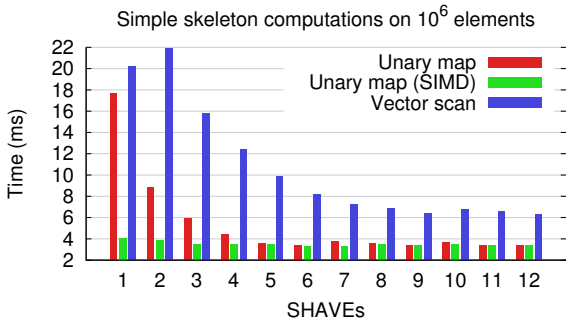

Fig. 2. Execution times for a simple Map skeleton, with and without SIMD, and a simple Scan skeleton, on Myriad2.

V. Experimental Results

Different benchmarks were run on a MV0182-R4 evaluation board, using a JTAG Debug Connector between a PC and the board to receive results. The Myriad2 chip was clocked at 600 MHz. For measuring time, one of the Myriad2's built-in timers was used.

A. Performance Scaling of SHAVE Cores

Figure 2 (red bars) shows the execution time of a SkePU test program containing a simple Map skeleton invocation with increasing number of SHAVE cores on Myriad2, where automatic SIMD vectorization was not used.

The time to load the program code to the SHAVEs is negligible when executing a single skeleton, in contrast to the previous solution with Myriad1 where SHAVE program load time (from host memory upon each skeleton invocation) was dominating [8]. This large reduction in code loading overhead is due to the new back-end design for Myriad2 that runs the

[2]Note from Movidius: Dynamic loading and symbol sharing is partially supported now and will be fully supported in MDK 15.12 December release.

[3]It turns out that inlining itself already has a big impact on performance. For instance, the performance of the Scan benchmark (see Section V) has improved even though it does not use SIMD. Also, having user functions virtual shows a noticeable performance penalty.

[4]DMA transfers could possibly be improved further by moving the handling of the transfers to a separate task to decrease DMA resource congestion.

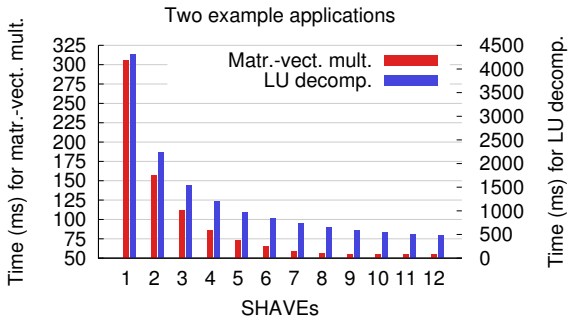

Fig. 3. Speedup for matrix-vector multiplication with a 4096x4096 matrix, and LU decomposition on a 256x256 matrix, on Myriad2.

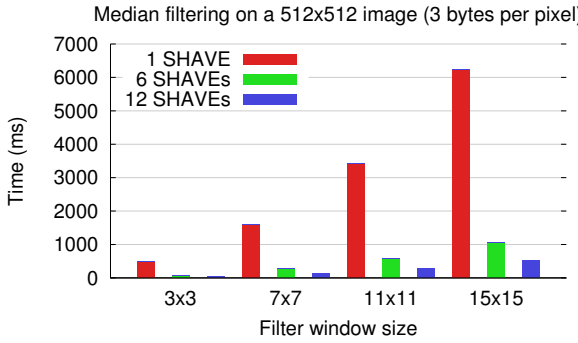

Fig. 4. Speedup for median filtering on Myriad2.

main program on a Leon core and thus eliminates the need to load code from the host.

In the case of the simple Map skeleton, using more than 7 SHAVE cores does not improve the execution time. The reason for this is that operands and results are streamed between the DDR memory and the CMX memory during the execution of the skeleton; this streaming becomes a bottleneck as the time spent doing actual calculations decreases. Using a more complex user function that performs more operations per element will show better performance improvement as the number of SHAVE cores is increased.

Note that the algorithm behavior changes for Scan when switching from 1 to multiple cores in Figure 2 (blue bars), as only one sweep over the operand data is necessary in the sequential case while the parallel algorithm needs to access each element twice.

Figure 3 shows the speedup of two test applications: Matrix-vector multiplication and LU decomposition. The latter uses many subsequent skeleton invocations, and in order to reduce the overhead of loading skeleton code to the SHAVEs, the skeleton code is kept loaded until a different skeleton is invoked. If two skeletons that use the same code (for instance unary Map) are called subsequently, the skeleton code is only loaded once, even if the operands or user function of the skeleton calls are different. In LU decomposition on a 256x256 matrix about 25 ms is saved.

Figure 4 shows the speedup of the median filtering example application included in the SkePU distribution, for four

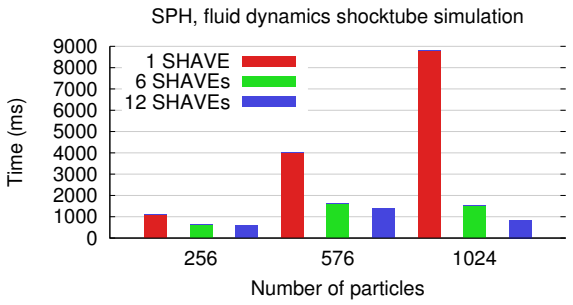

Fig. 5. Speedup for SPH fluid dynamics on Myriad2.

different filter window sizes.

Figure 5 shows the speedup of the Smooth Particle Hydrodynamics (SPH) fluid dynamics example application included in the SkePU distribution, for three different problem sizes.

B. Utilization of SIMD Features of SHAVE Cores

Figure 2 shows the impact of SIMD vectorization for the same Map skeleton on a 10^6 element vector. We can see that the optimization reduces execution times considerably; for a single-SHAVE scenario it speeds up execution by 4.4 times (18 to 4.1 ms). [5]

Figure 2 also shows execution times (blue) for a prefix sums program using the Scan skeleton, with inlining of the user function. Due to the loop-carried dependence structure, the SHAVE code created from the Scan skeleton could not be (automatically) vectorized. Inlining alone saves 3.6 ms here.

When using SIMD, the Map benchmark no longer scales after adding more than 3 SHAVE cores, due to that the streaming between the DDR memory and the CMX memory becomes a bottleneck. We verified this by disabling data transfers. The computational part of the skeleton scales almost perfectly when data transfers are disabled.

C. Overlapping (DMA) Data Transfer and Computations

Overlapping transfers improve performance of skeletons if a computationally intensive user function is used. If a simple user function is used, overlapping transfers makes skeleton execution a bit slower than synchronous transfers, due to overhead; a certain amount of time must be spent doing actual calculations for overlapping to be effective. On the other hand, if the computational intensity of the user function is very high, the performance gain of overlapping transfers becomes insignificant, as the time spent on data transfers remains constant with the size of the operands.

Currently a DMA buffer size of 12 kB is used. Larger buffers increase performance but leaves less room for other variables in the CMX memory.

[5] By using the compiler flag `-fno-vectorize`, we can disable auto-vectorizing to measure the speedup of using SIMD. The test program uses 32-bit operands, so that the vector arithmetic unit of each SHAVE core can perform computations on 4 operands at a time.

Application	Time [ms]	Power [W]	Eeff. [1/sW]
512x512 LU decomposition on PC	48.90	130.9	0.1562
512x512 LU decomposition on Myriad2	4963	0.6899	0.3624
SPH on 1024 particles on PC	177.4	103.8	0.05431
SPH on 1024 particles on Myriad2	843.5	0.6631	1.788
Dot product between two 10^7 long vectors on PC	0.8436	76.53	15.49
Dot product between two 10^7 long vectors on Myriad2	32.68	1.047	29.23

TABLE I
ENERGY EFFICIENCY COMPARISON BETWEEN PC AND MYRIAD2

D. Energy Efficiency Comparison

The power consumption of the Myriad2 chip, and of a PC with a GPGPU, was measured in three different SkePU example applications. The PC contains an Intel Xeon E5-2630L v2 CPU and an Nvidia Tesla K20c GPGPU. SkePU's CUDA backend was used on the PC. For the Myriad2, an MV0198 power measurement daughter card was used to measure the power consumption. For the PC system, the MeterPU [10] library was used to measure the aggregate power consumption of the CPU, DRAM, and GPGPU.

Table I shows the results from the measurements. Time, power, and energy efficiency are shown. Energy efficiency is calculated by $\frac{1}{time \cdot power}$. All three tested SkePU applications run slower, but more energy-efficiently on the Myriad2 than on the PC.

VI. RELATED WORK

To our knowledge, there is not much work done to port general-purpose skeleton programming frameworks to heterogeneous processors from the embedded computing domain.

BlockLib [11] is a skeleton framework for the Sony/-Toshiba/IBM Cell/B.E. (TM) heterogeneous multicore processor, which exposes a similar level of complexity to the programmer as the Myriad2 processor. In contrast to SkePU, BlockLib was simpler and based on C only, as there was no C++ support available for the slave processors on Cell/B.E.

Skell BE [12] is another skeleton framework for Cell, which implement skeletons as Embedded DSL by C++ metaprogramming, which removes some run-time overhead of polymorphism. The performance is comparable to handwritten code with a lightweight interface.

Compared to these frameworks, we offer, better support for execution on Myriad2, smart-container support to avoid unnecessary (implicit) data transfers between different skeleton invocations, which allows for optimization of program execution flow in a more global perspective.

VII. CONCLUSION

SkePU is designed for writing high-level and highly portable source code. Being a C++ include library, it is light-weight and does not rely on additional pre-compiler and analysis tools. In previous work on SkePU, we had shown that SkePU code efficiently executes on standard multicore CPUs and systems with single and multiple GPUs. In this work, we demonstrated the portability of SkePU programs to an architecturally very different processor type, the Movidius Myriad2 processor. We showed the feasibility of executing SkePU programs on a Myriad2 processor, hence verifying the generality of the skeleton programming concept on both HPC and embedded computing systems. We leveraged several Myriad2-specific optimizations such as vectorization and multi-buffering for overlapping data transfer with computations, to run SkePU programs on Myriad2 with decent performance.

ACKNOWLEDGMENT

Research partially funded by EU FP7 project EXCESS and SeRC project OpCoReS. We thank all EXCESS project members for comments on this work.

REFERENCES

[1] B. Barry, C. Brick, F. Connor, D. Donohoe, A. Lupas, S. Mitchell, D. Moloney, D. Nicholls, V. Toma, and R. Richmond, "Myriad2 "eye" of the computational-vision storm," in *26th Symposium on High Performance Chips (HotChips-2014)*, Aug. 2014.

[2] B. Barry, C. Brick, F. Connor, D. Donohoe, D. Moloney, R. Richmond, M. O'Riordan, and V. Toma, "Always-on vision processing unit for mobile applications," *IEEE Micro*, vol. 35, no. 2, pp. 56–66, Mar. 2015.

[3] M. I. Cole, *Algorithmic Skeletons : Structured Management of Parallel Computation*. Pitman Publishing, 1989. [Online]. Available: http://homepages.inf.ed.ac.uk/mic/Pubs/skeletonbook.pdf

[4] H. González-Vélez and M. Leyton, "A Survey of Algorithmic Skeleton Frameworks : High-Level Structured Parallel Programming Enablers," *Software-Practice & Experience - Focus on Selected PhD Literature Reviews in the Practical Aspects of Software Technology*, vol. 40, no. 12, pp. 1135–1160, 2010.

[5] J. Enmyren and C. W. Kessler, "SkePU: A multi-backend skeleton programming library for multi-GPU systems," in *Proc. 4th Int. Workshop on High-Level Parallel Programming and Applications (HLPP-2010), Baltimore, Maryland, USA*. ACM, Sep. 2010.

[6] S. Thorarensen, "A back-end for the SkePU skeleton programming library targeting the low-power multicore vision processor Myriad 2," Master's thesis, Linköping University, Sweden, 2016, to appear.

[7] U. Dastgeer and C. Kessler, "Smart containers and skeleton programming for GPU-based systems," *International Journal of Parallel Programming*, Mar. 2015, DOI: 10.1007/s10766-015-0357-6.

[8] R. Cuello, "Providing support for the Movidius Myriad1 platform in the SkePU skeleton programming framework," Master's thesis, Sep. 2014.

[9] *Movidius Compiler 00.50.62.0*, Movidius Ltd., 2015, technical Documentation.

[10] L. Li and C. Kessler, "MeterPU: A Generic Measurement Abstraction API Enabling Energy-tuned Skeleton Backend Selection." in *Proc. International Workshop on Reengineering for Parallelism in Heterogeneous Parallel Platforms (REPARA-2015) at ISPA-2015*, 2015, to appear.

[11] M. Ålind, M. Eriksson, and C. Kessler, "Blocklib: A skeleton library for Cell Broadband Engine," in *Proc. Int. Workshop on Multicore Software Engineering (IWMSE-2008) at ICSE-2008, Leipzig, Germany*. ACM, May 2008, pp. 7–14.

[12] T. Saidani, J. Falcou, C. Tadonki, L. Lacassagne, and D. Etiemble, "Algorithmic Skeletons within an Embedded Domain Specific Language for the CELL Processor," in *Parallel Architectures and Compilation Techniques, 2009. PACT '09. 18th International Conference on*, Sept 2009, pp. 67–76.