HArtes: Hardware-Software Codesign for Heterogeneous Multicore Platforms

Koen Bertels, Vlad-Mihai Sima, Yana Yankova, Georgi Kuzmanov, Wayne Luk, Gabriel Coutinho, Fabrizio Ferrandi, Christian Pilato, Marco Lattuada, Donatella Sciuto, Andrea Michelotti

Koen Bertels, Vlad-Mihai Sima, Yana Yankova, Georgi Kuzmanov, Wayne Luk, Gabriel Coutinho, Fabrizio Ferrandi, Christian Pilato, Marco Lattuada, Donatella Sciuto, and Andrea Michelotti. Hartes: Hardware-software codesign for heterogeneous multicore platforms. *IEEE Micro*, 30:88–97, September 2010

The final pubblication is available via http://dx.doi.org/10.1109/MM.2010.91

HArtes: Hardware-Software Codesign for Heterogeneous Multicore Platforms

Koen Bertels
Vlad-Mihai Sima
Yana Yankova
Georgi Kuzmanov
Delft University of Technology

Wayne Luk
Gabriel Coutinho
Imperial College London

Fabrizio Ferrandi
Christian Pilato
Marco Lattuada
Donatella Sciuto
Politecnico di Milano

Andrea Michelotti
Atmel Roma

Developing heterogeneous multicore platforms requires choosing the best hardware configuration for mapping the application, and modifying that application so that different parts execute on the most appropriate hardware component. The hArtes toolchain provides the option of automatic or semi-automatic support for this mapping. During test and validation on several computation-intensive applications, hArtes achieved substantial speedups and drastically reduced development times.

Multicore architectures can address memory, frequency, and power bottlenecks, while also providing a way around the impending limits of Moore's law that are rising over the CMOS horizon. In embedded systems, such multicore platforms are already widespread, and the heterogeneity of the processing elements (PEs) lets embedded-system developers design in a power-efficient way. However, one of the challenges is not only to map the applications efficiently on the available hardware components but also to determine which hardware components are necessary to satisfy the overall design objectives. This hardware-software codesign effort assumes it's possible to explore a large design space to find the best software mapping on the hardware. Such design space can be huge, especially when reconfigurable components are available that can support runtime reconfigurability. Moreover, without some kind of automation to explore various design choices, such an endeavor becomes rapidly intractable.

The hArtes (Holistic Approach to Reconfigurable Real-Time Embedded Systems) project's main objective is to develop an integrated toolchain that gives designers the option of automatic or semi-automatic support for the entire hardware-software codesign process. Although, in principle, there can be different design objectives such as low power or restricted bandwidth, the hArtes project assumes that application acceleration is one of the main objectives. The starting point is an existing or new application written in C, and the end point is an executable with a modified code mapped on a multicore platform. This platform consists of a general-purpose processor, a digital-signal processor (DSP), and a field-programmable gate array (FPGA).

Moreover, the hArtes toolchain embeds configuration bitstreams for the system's

Related Work on Mapping Applications to Multicore Heterogeneous Architectures

Researchers have developed various approaches for mapping applications to multicore heterogeneous architectures. Ptolemy is a research project from the University of California, Berkeley, that supports multiple computation models using a hierarchical heterogeneous design environment.[1] Ptolemy allows for designing and implementing complete, highly complex systems using different computation models at each hierarchical level, such as communicating sequential processes (CSPs) and synchronous data flows (SDFs), rather than being restricted by the generality of a homogeneous design environment.

However, this approach has three disadvantages compared to the hArtes approach. First, it requires expertise and knowledge about each computation model used. Second, it requires considerable manual effort to design the system. Third, porting components and functions manually or automatically from one processing-element type (such as an FPGA) to another (such as an ARM processor) is difficult with Ptolemy's approach.

Commercial tools from Compaan (http://www.compaandesign.com), CriticalBlue (http://www.criticalblue.com), and Impulse (http://www.impulseaccelerated.com) also target multicore applications. In particular, Compaan automatically maps computationally intensive streaming kernels to FPGAs. However its effectiveness is limited to specific application domains (stream based) and platforms (FPGA based). CriticalBlue's Prism and Cascade provide a thorough analysis of the application and a coprocessor synthesis on FPGAs, respectively. But CriticalBlue still requires manual effort to accelerate applications and is limited regarding the hardware platforms it can support. Impulse CoDeveloper supports hardware-software partitioning, but it's mainly suited for FPGA compilation. So, it has limited support for other heterogeneous architectures, such as those with digital-signal processors or graphics processing units (GPUs).

Reference

1. J. Eker et al, ''Taming Heterogeneity: The Ptolemy Approach,'' *Proc. IEEE,* vol. 91, no. 1, 2003, pp. 127-144.

reconfigurable components, thus providing a complete, operational system that's supported at both the software and hardware levels. The hArtes design flow restructures the code so that it's possible to exploit any available task-level parallelism. The toolchain then analyzes the restructured application to see whether and, if so, how these tasks can be mapped on particular hardware components. On the basis of the identified tasks, the implementation cost, and the developer's preferences, the toolchain selects the number of tasks for mapping. Various back-end compilers then insert the appropriate instructions that are necessary to start any of the available heterogeneous processing cores. A feedback loop ensures that certain choices can be evaluated and modified, after which the same design steps can be repeated. The benefits of the hArtes toolchain include the following:

- It uses a familiar programming paradigm.
- Platform complexity is hidden from the programmer.
- The tools can work with both new and legacy code.
- The approach allows a composable design in which, for example, it's possible to easily integrate already available IP blocks.
- The toolchain substantially reduces time to market and enables easy prototyping.
- It supports complete system generation at both the software and hardware levels.

The ''Related Work on Mapping Applications to Heterogeneous Multicore Architectures'' discusses other approaches in this area.

The Molen programming paradigm

The Molen programming paradigm is based on the Molen machine organization (see Figure 1), which defines how a general-purpose processor (GPP) interacts with one or more coprocessors. In the hArtes project, these coprocessors can be reconfigurable fabric or DSPs. But, in general, a coprocessor can be any other kind of processing core. There are two communication mechanisms between the GPP and the coprocessors: a shared memory and a dedicated register storage mechanism called the exchange register (XREG) file. The shared memory stores large pieces of data, letting all processing entities work with this data and thus reducing the need for costly memory transfers. XREGs are for transferring small pieces of data, such as function parameters, results, and pointers to locations in the shared memory, between the GPP and the coprocessors.

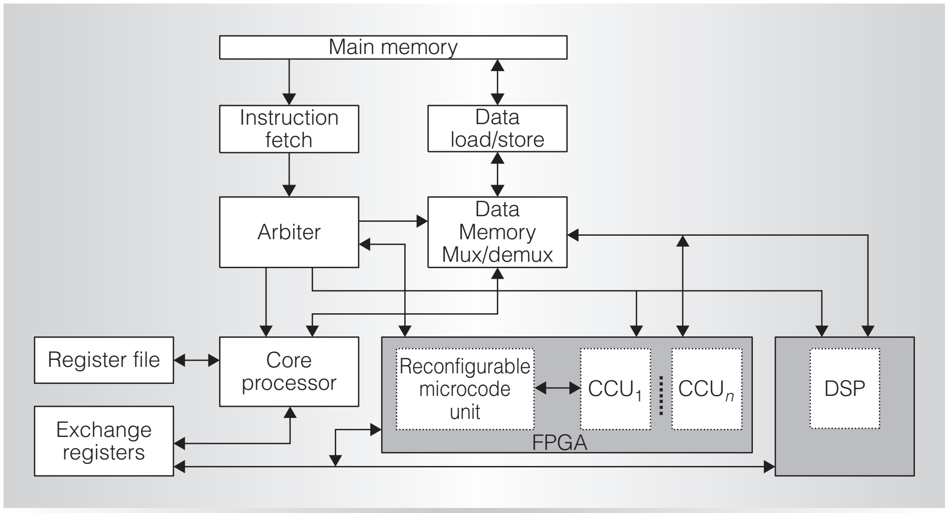
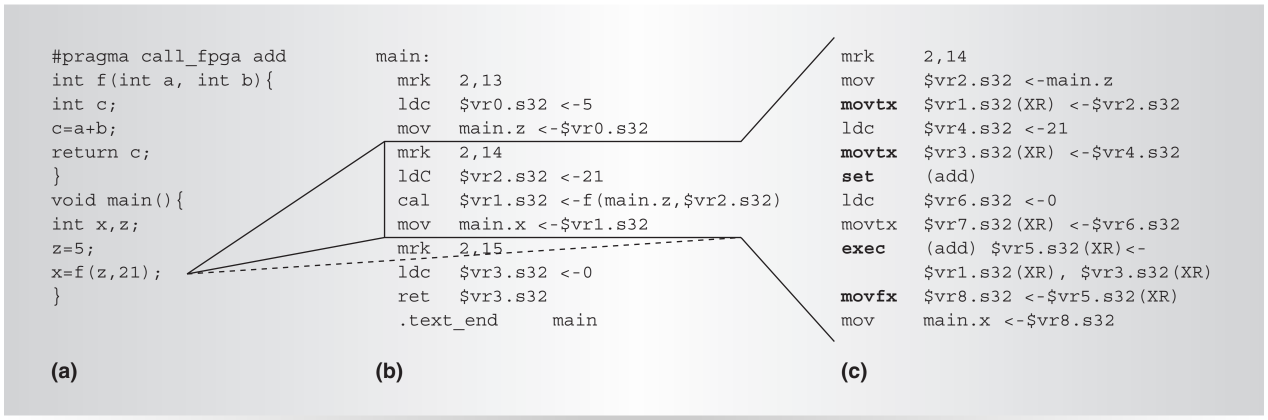


Figure 1. The Molen machine organization. The arbiter manages the interaction between the core processor and any number of custom computing units (CCUs) or digital-signal processors (DSPs) by partially decoding the core processor flow's instructions. (FPGA: field-programmable gate array.)

The Molen architecture involves a one-time extension of the instruction set architecture (ISA) to implement an arbitrary functionality.[1] For the hArtes project, we implemented an ISA extension of the following five instructions: `set`, `execute`, `movtx`, `movfx`, and `break`. These instructions make it possible to configure the FPGA or ensure that a coprocessor is ready (`set`); to start an operation execution on it (`execute`); or to exchange data via the exchange registers, moving to or from an XREG (`movtx`, `movfx`). The Molen programming paradigm is a sequential-consistency paradigm, in which we can partition an application such that certain parts run in parallel on the reconfigurable fabric while other parts run on the GPP. Furthermore, this paradigm has synchronization points, where all parallel threads come back together and the general execution flow continues.[1] This synchronization mechanism is supported by an explicit instruction (`break`). Although the Molen machine was initially proposed specifically for FPGA-based hardware platforms, it is also applicable to other kinds of coprocessor technologies. In the case of hArtes, the hardware platform consists of an ARM or PowerPC processor, an Atmel Diopsis DSP, a Xilinx Virtex-4, or an Altera Stratix II FPGA.

The Molen platform also features polymorphic program execution. As in any regular application, the instructions come from memory. But, rather than the GPP, an arbiter first partially decodes them. The arbiter decides whether an instruction is for the GPP or any of the available custom computing units (CCUs). In the hArtes case, the CCUs can be the DSP or any kernel mapped on the FPGA. The exchange registers transfer parameters to and from the CCUs. Data is also directly accessible by any of the CCUs in main memory. If the arbiter decodes a `set` instruction, the Molen architecture downloads the corresponding configuration bitstream to the FPGA, or sets up the DSP for particular functionalities. If the arbiter decodes an `execute` instruction, the arbiter initiates a CCU operation, and the CCU's control unit begins reading the memory pointers from the XREG, which address the main data memory. The arbiter performs a similar operation if an execution on the DSP is initiated.

At the software level, a set of pragmas annotate the program code (see Figure 2). We can clearly identify the position and sequence of the Molen-specific instructions in the modified assembly code generated by the Molen compiler. Essentially, a call to the hardware replaces the standard-operation assembly



```
#pragma call_fpga add
int f(int a, int b){
int c;
c=a+b;
return c;
}
void main(){
int x,z;
z=5;
x=f(z,21);
}
```

(a)

```
main:
  mrk   2,13
  ldc   $vr0.s32 <-5
  mov   main.z <-$vr0.s32
  mrk   2,14
  ldC   $vr2.s32 <-21
  cal   $vr1.s32 <-f(main.z,$vr2.s32)
  mov   main.x <-$vr1.s32
  mrk   2,15
  ldc   $vr3.s32 <-0
  ret   $vr3.s32
  .text_end     main
```

(b)

```
mrk     2,14
mov     $vr2.s32 <-main.z
movtx   $vr1.s32(XR) <-$vr2.s32
ldc     $vr4.s32 <-21
movtx   $vr3.s32(XR) <-$vr4.s32
set     (add)
ldc     $vr6.s32 <-0
movtx   $vr7.s32(XR) <-$vr6.s32
exec    (add) $vr5.s32(XR)<-
        $vr1.s32(XR), $vr3.s32(XR)
movfx   $vr8.s32 <-$vr5.s32(XR)
mov     main.x <-$vr8.s32
```

(c)

Figure 2. The Molen programming paradigm: C code (a), intermediate code (b), and modified intermediate code (c). The function code in the original C program is translated in a modified intermediate representation.

code, executing the same operation either on a reconfigurable CCU or (in hArtes) on a DSP.

The hArtes toolchain

The goal of the hArtes toolchain is to let developers write their applications at a software level and exploit the latest advances of heterogeneous platforms without expert low-level hardware knowledge. The toolchain achieves this by

- giving designers the option of automatic or semi-automatic support such that, at each toolchain level, the tools introduce architectural decisions, which developers can override; and
- abstracting low-level details through source annotations.

Source annotations are introduced in the application by C pragmas, which are used primarily to describe parallelism and mapping by either developers or tools.

The hArtes toolchain uses several pragma annotations, including `#pragma omp`, `#pragma map`, `#pragma profile`, and `#pragma issue`, which we describe in more detail in the following. There are also ways to use these pragmas so that designers can force a particular mapping. Thus, it's possible to use the tool in an automatic or semi-automatic way.

The `#pragma omp` pragma annotation

OpenMP pragmas specify where parts of the application can execute in parallel. Each section can thus execute independently of, and in parallel with, the other sections in the defined OMP parallel section, as in the following example:

```
#pragma omp parallel sections
{
   #pragma omp section
   {
     FFT();
   }
#pragma omp section
   {
     DCT();
   }
}
```

The `#pragma map` pragma annotation

This pragma can be used on top of a function declaration or a specific function call to indicate that a particular hardware component (such as a DSP or an FPGA) should be used instead. This source annotation has two parameters: the component name (such as `MAGIC` in the following example, referencing the DSP) and the specific implementation identification number associated with that component:

```
#pragma map call_hw MAGIC 1
void funcA(int *p);  //all calls to
  funcA will be executed in the
  mAgicV DSP
...
{
   #pragma map call_hw VIRTEX4 2
   funcB(x);  //this particular
        call will be executed in the
        Virtex4 FPGA
}
```

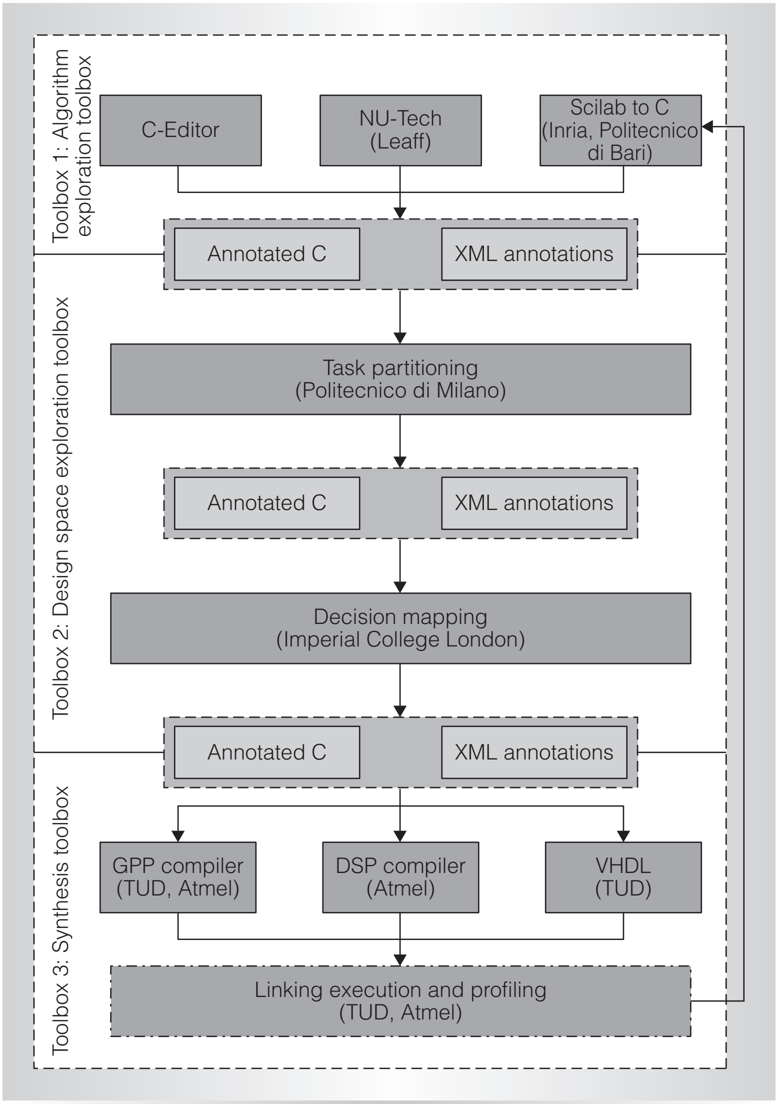

Toolbox 1: Algorithm exploration toolbox
C-Editor
NU-Tech (Leaff)
Scilab to C (Inria, Politecnico di Bari)
Annotated C
XML annotations
Toolbox 2: Design space exploration toolbox
Task partitioning (Politecnico di Milano)
Annotated C
XML annotations
Decision mapping (Imperial College London)
Annotated C
XML annotations
Toolbox 3: Synthesis toolbox
GPP compiler (TUD, Atmel)
DSP compiler (Atmel)
VHDL (TUD)
Linking execution and profiling (TUD, Atmel)

Figure 3. The hArtes toolchain. Each toolbox processes C code in accordance with inline annotations and XML annotations. The final result is a single file consisting of linked binary executables and configuration bitstreams. [GPP: general-purpose processor; Inria: Institut National de Recherche en Informatique et en Automatique (National Institute for Research in Computer Science and Control); TUD: Technische Universiteit Delft (Delft University of Technology).]

The `#pragma profile` pragma annotation

This annotation gives mapper-profiling information on, for instance, the CPU time consumed by a particular function or the number of times the function is called. In the following example, the pragma is a tuple containing the function's mean execution time and the number of times it's called, as well as the corresponding variance of those two mean values:

```
#pragma profile time(30,0.02)
  num_calls(5,0.8)
void funcB(int *p);
```

The `#pragma issue` pragma annotation

This annotation provides general information about an application. Thus far, we've used only a "black box" type annotation that forces routines to be excluded from the partitioning process, as in the following example:

```
#pragma issue blackbox
void funcC(int *p);
```

The three toolboxes in hArtes toolchain

The hArtes toolchain contains three toolboxes, each taking annotated C code and XML annotations as its input. The original application C code is annotated inline and processed according to XML annotations by the tools in a given toolbox. The resulting C code and modified XML annotations are conveyed to the next toolbox. Figure 3 gives a high-level overview of the entire toolchain.

At the toolchain's top level is the *algorithm exploration toolbox* (AET), which contains tools for developing and refining algorithms at a higher abstraction level. This toolbox includes Leaff's NU-Tech tool (http://www.nutech.com) and the open-source tool Scilab (http://www.scilab.org), developed by Inria (France's National Institute for Research in Computer Science and Control) and Politecnico di Bari. NU-Tech is a graphical software development tool that lets developers graphically design and simulate their application by connecting logic blocks together, after which C code is generated. Scilab is the open source equivalent to Matlab. Scilab takes Matlab code as input and then generates a C version of it. Use of the AET is optional; developers can write C code directly as an entry point to the toolchain.

The next level of the toolchain is the *design space exploration (DSE) toolbox*, where the application undergoes various transformations. The DSE toolbox consists of two processes: partitioning and mapping.

The partitioning tool is named Zebu, and it is developed inside PandA, the hardware-software codesign framework currently under development at Politecnico di Milano. Taking into account information on each solution's potential speedup, Zebu identifies the tasks, at the proper granularity, in which the application can be decomposed to improve its performance. Zebu starts from the application's C code and takes into account information extracted from the XML file containing the platform specification. Through a feedback loop, Zebu also takes into account information regarding the possible acceleration of certain tasks. Zebu behaves as a compiler in that it consists of

- a front end, which creates the intermediate representation of the input code using a slightly modified version of the GNU compiler (GCC);
- a middle end—the Zebu core—where the tool creates an efficient partitioning,[2] exploiting internal performance estimation techniques,[3] and then performs transformations on the resulting task graph to determine the tasks' proper granularity in order to account for the additional overhead required to manage them; and
- a back end, which generates the executable C code, annotated with OpenMP and mapping directives.

Subsequently, the hArmonic mapping tool (developed by Imperial College London) selects parts of the C application to execute on specialized processors (DSPs or FPGAs), to accelerate the application. The mapping process provides two novel features.[4] First, it's possible to rapidly generate near-optimal solutions by combining an inference rule engine (which generates solutions by construction from any point of design space) with a heuristic search algorithm. Second, application developers can influence the task-mapping process by providing directives to guide the search for a mapping solution. The mapping tool accepts as input an arbitrary number of C source files and the XML platform specification. The mapping process then automatically verifies whether established C guidelines are satisfied, and it pinpoints potential problems in the source that may prevent a good or feasible mapping. Following this stage, the mapping process determines, on the basis of a set of filtering rules, which PEs can support each part of the application to minimize the mapping search space and comply with the hArtes platform's requirements and limitations.

The final stage of the toolchain is the *synthesis toolbox*, developed by Delft University of Technology and Atmel, which contains all the necessary back-end C compilers for each PE type (such as GPPs, DSPs, and FPGAs) in the system. Once hArmonic provides a mapping solution,[5] it generates C source code for each compiler in the synthesis toolbox. Each source is compiled separately, and then linked together to form one single binary, which contains heterogeneous object code from different PEs. This binary, which the hArtes runtime interface supports, provides the following services: platform and PE initialization, remote procedure call (Molen programming model) and remote thread control, memory allocation, profiling, and debugging support.

The GPP compiler, which is based on GCC 4.3, generates the Molen instructions for the annotated functions. It also does a basic scheduling of the `execute` and `break` Molen calls needed to parallelize the procedure calls found in OpenMP sections. The hArtes project implements the Molen instructions as runtime library function calls to even further ease the addition of new architectures.

The FPGA compiler is the Delft Workbench Automated Reconfigurable VHDL Generator (Dwarv),[6] which is a C-to-VHDL generation toolset. This compiler exploits the available parallelism of algorithms and generates designs suitable for hardware-software coexecution using the Molen programming paradigm. It consists of two modules: the DFG (Data-Flow Graph) Builder and the VHDL Generator. The DFG Builder is based on the Stanford University Intermediate Format (SUIF) compiler infrastructure and has as input a standard C source, from which it generates a hierarchical DFG.

Table 1. Kernel-only acceleration for noise reduction in an ARM processor, a digital-signal processor (DSP), and an FPGA (Xilinx Virtex-4).

Execution	ARM	DSP	FPGA (Xilinx Virtex-4)
FFT kernel (μs)	13,055	3,261	1,843
Speedup	1	4	7

The VHDL Generator takes the DFG and, using an "as soon as possible" scheduling, generates synthesizable VHDL. When needed, Dwarv can use already-available IP blocks. A good example is the floating-point library, which is based on vendor-specific designs such as Xilinx IP, and which Dwarv uses when compiling C code that contains floating-point operations.

The hArtes linker produces a single executable that links all the contributions from the different PE compilation chains. These contributions include executable binaries and configuration bitstreams. The linker is based on the GNU linker (LD) and is targeted for ARM Linux and a customized hArtes linker script. The bitstreams and the DSP executable files are transformed accordingly so that they can be integrated into the ARM executable and linkable format (ELF) file.

Technical validation

To validate the Molen computational paradigm and assess the current available toolset, we mapped different applications on the hArtes hardware platform. Here, we don't elaborate on all the design process stages; we simply report on the results obtained. All speedups reported here are relative to the execution of the application on the hArtes board, where the ARM processor provided the baseline. We then generated mappings on either the DSP or the FPGA, and we report the speedup results when executing the application on the ARM with the DSP, or on the ARM with the FPGA. It's important to stress that we derived the numbers reported here from real execution of the entire application, not simulations. Finally, even though the entire approach lets us execute multiple kernels in parallel according to OpenMP annotations, two reasons prevented us from real parallel execution. The first is the limited area available for mapping large kernels. The second is related to the way multiple functions are combined into one hardware kernel. When multiple functions are mapped on the FPGA, the Dwarv hardware compiler combines all those functions' DFGs to minimize execution time, but also to reuse available hardware blocks when possible.

Wave-field synthesis

One of the applications we used to validate the toolchain was a wave-field synthesis application. This computation-intensive audio application used multiple audio inputs and outputs to compute the spatial properties of sound sources in an immersive environment such as a room or a car. A fully automatic use of the toolchain resulted in a mapping of 15 functions to the DSP, reaching an overall application speedup of 9.24 compared to an ARM-based execution.

Noise filter

Another application tested was a noise filter. This application contained a down sampler and an up sampler, fast Fourier transforms (FFTs), and several other audio-processing routines. For this application, the toolchain automatically mapped 16 functions to the DSP and 10 functions to the FPGA. The cost estimator, which predicts how an arbitrary function performs in each PE, identified the FFT kernel as a candidate for acceleration—a result which the dynamic profiler confirmed.

As Table 1 shows, the kernel-only execution of the FFT (without taking into account communication costs between PEs) on the DSP resulted in a speedup of 4, versus 7 for the Xilinx Virtex-4 FPGA. However, the hArtes board has bandwidth restrictions between the FPGA and external memory, resulting in a 2.7 overall speedup for the entire application (see Table 2). When we applied manual code modifications affecting memory management, the FPGA-based mapping's performance reached a 3.1 speedup. On the other hand, when we mapped the FFT on the DSP, the overall speedup was 4.5.

The ARM processor alone cannot meet the real-time requirement for this application—that is, process 256 samples/cycle in less than 16 ms. But using the toolchain with a complete automated approach and without any additional code modification achieved this real-time performance requirement, with a substantial reduction in overall development time as well.

User validation

An equally important objective of the hArtes project is to assess to what extent a toolchain such as the one described here can reduce time to market and simplify the entire design process. To reach this objective, we asked our application provider partners to conduct a validation, making a qualitative and partly quantitative comparison of the design effort with and without the toolchain. A dramatic reduction of the mapping effort was evident from the best experiences in this exercise. The entire toolchain analysis took only a few minutes, and the most time-consuming parts were typically proprietary synthesis tools—for example, from the FPGA vendors.

In addition, we organized an international design contest to get feedback from non-hArtes partners. The design contest involved mapping an application of their choice to the hArtes board (the DSP, in this case) using the hArtes tools. A total of 11 teams from all over the world enrolled in the contest and provided their solutions. We asked the teams to describe their experiences and assess how much the tools facilitated the entire development effort. The chosen application domains were diverse, ranging from solar-panels management to a mobile Web identification and location system, to a satellite communication system. From this validation exercise, along with the experiences of our application provider partners, the following benefits of the hArtes toolchain are apparent:

- *Familiar programming paradigm.* Developers can program in the same way as usual and make a simple annotation, either manually or using the tools, to indicate where and how to accelerate the program execution. For the developer, using an accelerator is as simple as calling a function. Especially in the case of FPGA-based acceleration, the automatic generation of VHDL code has proved to be of extreme importance because it's a known development bottleneck for this technology.
- *Hiding platform complexity.* The developer need not understand platform complexity, nor even what the exact hardware components are.
- *Flexible approach for both new and legacy code.* The toolchain is indifferent to whether developers want to port existing code to such platforms or build it from scratch.
- *Composable approach allowing for integration of any available IP blocks.* If developers dispose of previously built or even purchased IP blocks, the hArtes approach allows them to easily integrate these IP blocks in the application.
- *Substantial reduction in time to market and easy prototyping.* Not only the contest participants but also the hArtes application partners experienced substantial time-to-market reduction. For example, the noise filter application was compiled to the hArtes board in a matter of minutes, whereas a manual mapping would have taken multiple days.

Table 2. Processing time for one application cycle of noise reduction after mapping the original application code on an ARM processor, a DSP, and a Xilinx Virtex-4 FPGA; and after mapping a manually tuned application code on a Xilinx Virtex-4 FPGA.

	Original application code			Manually tuned application code
Execution	ARM	DSP	FPGA (Xilinx Virtex-4)	FPGA (Xilinx Virtex-4)
Noise filter application (μs)	27,000	6,000	10,000	8,600
Speedup	1	4.5	2.7	3.1

The current status of the hArtes toolchain is not yet of industry-grade quality, nor has it reached an end point in terms of mapping functionality. Reaching that stage of maturity and improved functionality will require several improvements.

First, we must improve the hardware-software codesign support. Although the current approach takes the target hardware platform as given, it's possible in principle to scale the hArtes board by adding yet another, identical heterogeneous tile. We must also extend the partitioning and mapping algorithms so that they also suggest the most appropriate number of processing units necessary to satisfy the design objective.

Second, we need to relax some limitations of the Dwarv hardware compiler. Several restrictions imposed on the C language, such as recursion, will never be supported by the Dwarv hardware complier. But others, such as structures, are just a matter of the appropriate development effort.

Third, to improve development support even further, the programming environment should provide, early in the development process, additional coding recommendations and profiling information.

Fourth, we need to explore memory-mapping and memory-transfer optimizations. Memory bandwidth is a well-known bottleneck for multicore platforms with shared memory. More sophisticated code analysis techniques and corresponding optimizations are necessary.[7]

Fifth, we must integrate advanced debugger facilities. A feature such as stepwise debugging of an FPGA-based kernel execution isn't a realistic option in the near future. However, a similar functionality in which a DSP compiler provides the necessary debugging information in an accessible format might be realistic. Finally, we need to restructure the application source code to support efficient execution of a specific PE. For example, certain optimizations could compensate for a lack of bandwidth, or we could improve the use of local buffers by inlining FPGA-mapped functions.

To address these challenges, a core team from the hArtes project derived a company called BlueBee Multicore Technologies, with a first release of the new BlueBee toolchain targeted for 2011. MICRO

Acknowledgments

This research has been funded by the hArtes project (EU-IST-035143).

References

1. S. Vassiliadis et al., ''The Molen Polymorphic Processor,'' *IEEE Trans. Computer,* vol. 53, no. 11, 2004, pp. 1363-1375.
2. F. Ferrandi et al., ''Automatic Parallelization of Sequential Specifications for Symmetric MPSoCs,'' *Proc. Int'l Embedded Systems Symp.* (IESS 07), IFIP 231, Springer, 2007, pp. 179-192.
3. F. Ferrandi et al., ''Performance Estimation for Task Graphs Combining Sequential Path Profiling and Control Dependence Regions,'' *Proc. 7th ACM/IEEE Int'l Conf. Formal Methods and Models for Codesign* (Memocode 09), IEEE Press, 2009, pp. 131-140.
4. W. Luk et al., ''A High-Level Compilation Toolchain for Heterogeneous Systems,'' *Proc. IEEE Int'l SoC Conf.* (SOCC 09), IEEE Press, 2009, pp. 9-18.
5. Y.M. Lam et al., ''Mapping and Scheduling with Task Clustering for Heterogeneous Computing Systems,'' *Proc. Int'l Conf. Field Programmable Logic and Applications* (FPL 08), IEEE Press, 2008, pp. 275-280.
6. Y.D. Yankova et al., ''DWARV: DelftWorkbench Automated Reconfigurable VHDL Generator,'' *Proc. Int'l Conf. Field Programmable Logic and Applications* (FPL 07), IEEE Press, 2007, pp. 697-701.
7. V.-M. Sima and K. Bertels, ''Runtime Memory Allocation in a Heterogeneous Reconfigurable Platform,'' *Proc. Int'l Conf. Reconfigurable Computing and FPGAs,* IEEE CS Press, 2009, pp. 71-76.

Koen Bertels is an associate professor in the Computer Engineering Lab, at Delft University of Technology. His research interests include electronic system-level design, reconfigurable computing, and multicore computing. Bertels has a PhD in computer information systems from the University of Antwerp. He is a member of IEEE and the IEEE Computer Society.

Vlad-Mihai Sima is pursuing a PhD in computer engineering at Delft University of Technology. His research interests include compilers, reconfigurable computing, and heterogeneous embedded systems. Sima has an MSc in computer science and engineering from Universitatea Politehnica in Bucharest, Romania.

Yana Yankova is pursuing a PhD in computer engineering at Delft University of Technology. Her research interests include compilation systems, automated hardware generation, reconfigurable computing, and embedded systems. Yankova has an MSc in computer systems from the Technical University of Sofia in Bulgaria.

Georgi Kuzmanov is an assistant professor in the Computer Engineering Lab at Delft University of Technology. His research interests include reconfigurable computing, media processing, computer architecture and organization, and embedded systems. Kuzmanov has a PhD in computer engineering from Delft University of Technology. He is a member of IEEE and the IEEE Computer Society.

Wayne Luk is a professor of computer engineering at Imperial College London. His research focuses on theory and practice of customizing hardware and software for specific application domains, such as multimedia, financial modeling, and medical computing. Luk has a DPhil in engineering and computing science from the University of Oxford. He is an IEEE Fellow.

Gabriel Coutinho is a research associate in the Custom Computing Research Group of the Computing Department at Imperial College London. His research interests include hardware compilation, source-level transformations, high-performance computing, and embedded systems. Coutinho has a PhD in computer science from Imperial College London.

Fabrizio Ferrandi is an associate professor in the Department of Electronics and Information at Politecnico di Milano. His research interests include synthesis, verification simulation, and testing of digital circuits and systems. Ferrandi has a PhD in information and automation engineering from Politecnico di Milano. He is a member of IEEE and the IEEE Computer Society.

Christian Pilato is pursuing a PhD in information technology at Politecnico di Milano. His research interests include high-level synthesis, evolutionary algorithms for design space exploration and multiobjective optimization, and multiprocessor design. Pilato has an MSc in computing systems engineering from Politecnico di Milano.

Marco Lattuada is pursuing a PhD in information technology at Politecnico di Milano. His research interests include automatic parallelization of sequential specification, high-level synthesis, and integer linear programming. Lattuada has an MSc in computing systems engineering from Politecnico di Milano.

Donatella Sciuto is a full professor in the Department of Electronics and Information at Politecnico di Milano. Her research interests include embedded systems and multicore systems, including reconfigurable and adaptive systems. Sciuto has a PhD in electrical and computer engineering from the University of Colorado, Boulder. She is a member of IEEE.

Andrea Michelotti is a hardware-software codesign manager at Atmel Roma. His technical interests include DSP (digital-signal processor) hardware and software design, embedded operating systems, system integration, and test vector generation. Michelotti has an MSc in physics from the Sapienza University of Rome.

Direct questions and comments about this article to Koen Bertels; Computer Engineering Lab; Faculty of Electrical Engineering, Mathematics, and Computer Science; Delft Univ. of Technology; Mekelweg 4; 2628 CD Delft; the Netherlands; k.l.m.bertels@tudelft.nl.