PHAT: A TECHNOLOGY FOR PROTOTYPING PARALLEL HETEROGENEOUS ARCHITECTURES

Thorsten Wink, Andreas Koch

Embedded Systems and Applications Group
Technische Universität Darmstadt
email: {wink|koch}@esa.cs.tu-darmstadt.de

ABSTRACT

This paper presents the Parallel Heterogeneous Architecture Technology (PHAT), a scalable design methodology for prototyping and evaluating heterogeneous arrays of software-programmable VLIW processors and both manually designed and automatically-compiled custom hardware accelerators, using a shared memory architecture for communication. We discuss the trade-offs and break-even point for switching from bus-based to network-on-chip interconnects, the interface and protocols for connecting distributed on-chip caches and multi-bank out-of-order off-chip-memories, as well as the impact of floorplanning on the quality of results for implementation on Xilinx Virtex 6 LX 760 devices.

The capabilities are evaluated at the system-level on the multi-FPGA Convey HC-1ex hybrid-core computer, accessing its high-performance memory system, and integrating r-VEX processor cores with IP blocks for SHA and FFT computations.

I. INTRODUCTION

From application domains reaching from high-performance stationary to mobile or embedded computing, single monolithic processors are being replaced with heterogeneous arrays of processing elements, combining both programmable and fixed-function as well as (in some cases) reconfigurable components. Often, shared memory is used as the interaction model between the different processing elements as explicit message-based communication requires a higher effort from the software developers.

Such systems are becoming increasingly difficult to develop and prototype: Since custom hardware blocks (hard IP blocks or custom-compiled components) often need to be considered, fast simulation using just instruction-set or architectural simulators is no longer sufficient. While it is possible to combine different simulators (instruction-set and RTL), the performance of such a system is often sub-optimal due to numerous cross-simulator calls, e.g., when modeling shared memories or cache-coherency mechanisms.

As an alternative, we present PHAT, a highly flexible framework for *emulating* parallel heterogeneous computing systems. In contrast to many conventional ASIC emulation approaches, our platform integration will provide both a fully integrated desktop-class processor as well as high-throughput memory for the processing elements (PEs) in the heterogeneous computing array. Note that we use the term PE to include both software-programmable processors as well as pure hardware IP blocks.

For flexibility, we support heterogeneity on a wide number of axes: Instruction set-extension and variable architecture parameters for individual software-programmable processors, integration of manually designed or automatically compiled hardware accelerators, a common but highly adaptable memory system, and an easily resizeable floorplanning scheme to provide physical-level support for high performance.

The technology described here combines prior work on configurable processors [20] and networks-on-chip [17] with our own research on configurable memory systems [15] and hardware/software co-compilers [10]. We will evaluate the system using combinations of pure software as well as IP and custom-compiled blocks.

II. RELATED WORK

With the growing interest in heterogeneous multiprocessors, described, e.g., in [11], demand for quicker design space exploration [12] [7] has also increased. For homogeneous multiprocessors, projects such as RAMP Gold [1] provide simulation accelerators (e.g., for 64 SPARC V8 cores communicating by shared memory), but do not allow the flexible integration of different PEs. PHAT aims to offer this capability to designers, enabling high-level architectural flexibility while optimizing physical-level issues for high performance. HASim [18] is another simulation system implemented on an FPGA. It can model a shared memory multicore system, using time multiplexing hardware to accelerate parts of the system. FAST [2] is a combined hardware / software simulator, where the functional simulation is done in software and only the timing model is realized in hardware. ProtoFlex [4] is a hybrid full system simulator used in the BlueSPARC simulator.

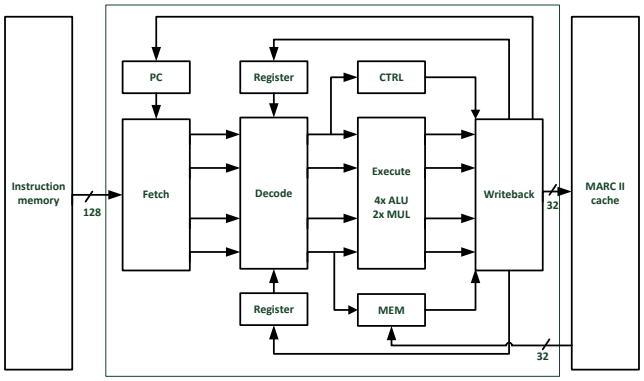

Fig. 1. r-VEX in a 4-issue configuration

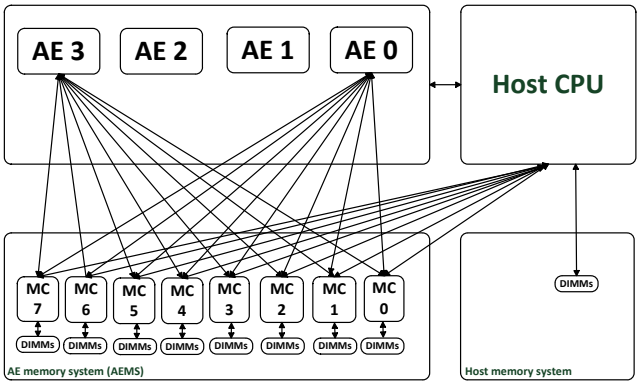

Fig. 2. Convey HC-1ex System

III. R-VEX

For many experiments, it is useful to easily alter the characteristics of the software-programmable processor. Common choices vary the types and numbers of processing units, while in some cases it is also desirable to extend the base instruction set with application-specific functionality.

One of the few openly available cores that provides the desired flexibility is r-VEX, developed by TU Delft [20]. It is an open implementation of the HP VEX architecture [5], of which commercial variants were used, e.g., in the ST Microelectronics ST200 VLIW-DSP processors.

Shown in Figure 1, r-VEX is a 32b VLIW micro architecture easily adaptable, e.g., with regard to the issue width, number of ALUs, Multipliers, Load and Store Units and many more parameters. By setting the issue width to one, it can be turned into a conventional RISC processor. The flexible architecture is supported by a configurable compiler tool chain [8] that generates code for the specific micro architecture instance. Additionally, since the core implementation is provided as synthesizable HDL, custom instructions can also be added to each instance as required.

In the original r-VEX distribution, a core could only operate in stand-alone fashion, using dedicated on-chip memories for instructions and data. Memory contents could only be initialized at synthesis time and no formalized communication scheme to the outside world existed. We have made numerous extensions to the base architecture: Accessing shared memory, configurable cache support for data and instructions, and interfacing with an external control processor for system management (including loading of programs and data) and I/O.

The specific r-VEX configuration we use here employs four ALUs, two multipliers, and a single load-store unit. Since this paper focuses on the system-level aspects, we will not discuss the instruction set-extension of individual processors here.

IV. INITIAL TARGET PLATFORM: CONVEY HC-1EX

While PHAT is designed for portability between platforms (discussed in Section X), our initial choice of platform is motivated by our interest in evaluating systems combining a powerful desktop/server-class CPU with shared memory heterogeneous processing arrays. Especially crucial is providing the heterogeneous array with sufficient memory bandwidth, a capability that is only rarely available on ASIC prototyping or FPGA evaluation boards.

Thus, as first implementation platform, we picked a Convey HC-1ex heterogeneous ("hybrid") computing system. The Convey HC-1ex system consists of a host server providing an Intel Xeon L5408 Quad-Core CPU, which is linked by FSB to a reconfigurable component consisting of four Application Engines (AE), each a Xilinx Virtex 6 LX760 FPGA. The system maps the host memory and the eight memory banks directly available to the AEs into a shared cache-coherent address space with NUMA characteristics (accesses from CPU to AE memory are possible, but slower than local accesses). The CPU controls the configuration of the AEs and can also remap parts of the virtual address space between CPU and AE memory for faster access. In this fashion, a seamless computation model similar to the one proposed earlier in [13] is achieved, which freely allows passing pointers between software and hardware execution.

Since the multiple compute units on the heterogeneous arrays have correspondingly higher memory bandwidth requirements, the powerful AE Memory System (AEMS) of the Convey platform is attractive for our-use case. In addition to the host memory shared via FSB, the 48b adress space allows the AEs access to eight banks of on-board memory, each bank handled by its own Memory Controller (MC) in a dedicated FPGA (saving space in the AEs). Each of the memory banks is 8 bytes wide and allows configurable interleaving of the address space. In the experiments presented in Section VIII, we have used linear interleaving which stores successive 8B blocks to

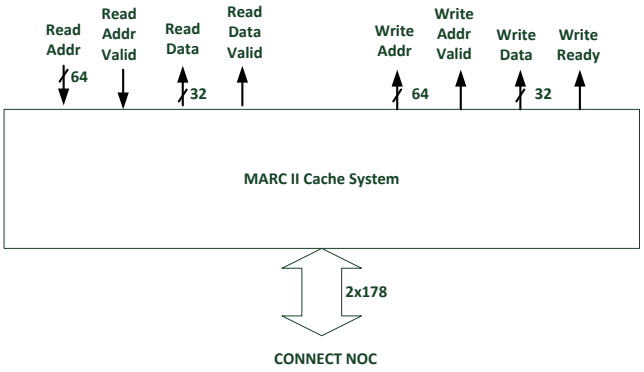

Fig. 3. External view of a cached configuration of the MARC II memory system with one read and write port (R1W1)

successive memory banks. While the AEMS aggressively exploits out-of-order operation for high throughput, the clients (processors and hardware accelerators) are responsible for efficiently reordering the incoming access replies (see Section V) using a 32b read ID field.

Typically, the Convey platform would be programmed either manually or using a hardware/software co-compiler such as Nymble [10] to execute the compute intensive parts of an application on the AEs and leave the rest of the code (management, I/O) on the Xeon host processor. With PHAT, more levels of partitioning become available: Not only between host CPU and AEs, but also between the (possibly differently configured) r-VEX cores and heterogeneous accelerators.

V. PROCESSING ELEMENT-LOCAL MEMORY INTERFACE

Since all of our PEs are capable of autonomously performing memory accesses (are master-mode capable), the memory interface in each PE to the shared system memory (both AE- and CPU-side) plays a key role in system performance.

The PE-local Memory Interface (PEMI) consists of two sides, one facing its client PE, the second facing the on-chip interconnect (see Section VI). Both of these sides use standardized protocols, so they can easily be applied to different PEs or connected to different communications mechanisms.

The differences lie in the configurable behavior of the PEMI: PEs such as processors and some accelerators profit from local caching, to avoid accessing the high throughput but also high latency off-chip AEMS. Other PEs perform streaming accesses and do not require caches, but still need the reordering capabilities required to receive data from the Convey AEMS.

To satisfy these different needs, the PEMI can provide either a simple reordering buffer for streaming accesses, or a complete configurable multi-port cache for more random access patterns. For the latter, we use a variation of our earlier MARC II [15] memory system. MARC itself is also highly portable, carefully separating the client-facing front-ends (shown as an example in Figure 3) from the technology-specific back-ends. In addition to the existing MARC back-ends for various kinds of SDRAM, SRAM, and PCI, a new back-end for interfacing with the AEMS was integrated. The mid-end of MARC is responsible for the actual caching behavior (direct-mapped/associative, write-through/write-back, cache organization, prefetching etc.). It was here where we added the reordering capability required to accept replies from the AEMS, the actual reordering is completely transparent for the client PE.

Since all of the heterogeneous elements have their own MARC instances, thrashing or cache pollution is completely avoided. MARC is also capable of supporting multiple levels of cache-coherency, but this feature was not yet exploited for the work presented here. For the actual experiments, we will use MARC caches, each configured with one read and one write port, direct mapped access, write-back mode, using a line width of 8 bytes, 1024 lines total, and a prefetch length of 32 lines.

To better support the r-VEX cores, we added another capability to the PEMI: The Fetch stage of the cores is directly attached to a dedicated local instruction memory (IMEM), consisting of fast on-chip BlockRAMs. When starting an r-VEX PE, that memory is initialized with the program to be executed from the shared memory using the PE's MARC instance (similar to the approach used in [6]). Afterwards, the MARC instance will only be used for data accesses. This setup essentially doubles the total memory bandwidth (instructions and data) for each r-VEX core. In the current experiments, the IMEM is configured to hold 1024 instructions, each 128b wide for a 4-issue r-VEX. This suffices to hold the code of the benchmark examples and could easily be extended for larger programs. Also, r-VEX itself needed to be modified to accept variable-latency memory accesses, as the original micro-architecture relying on local BlockRAMs always expected single cycle accesses. While IMEM can maintain this for instructions, cache misses for data accesses will now stall the entire r-VEX core until the data has been retrieved from main memory.

VI. ON-CHIP COMMUNICATION

In our previous work on adaptive reconfigurable computers such as [14], we were able to efficiently use custom bus-based interconnects between the different blocks (e.g., processor, accelerator(s), memory controller(s), ...). Thus, we initially attempted to apply this approach to the PHAT prototype. As each MARC instance can be configured to deal with multiple back-ends (mapped into different ranges of the address space), we created a separate MARC back-end for each of the eight Convey MC Interfaces (MCI) in each PE-specific MARC instance. At each MCI, local

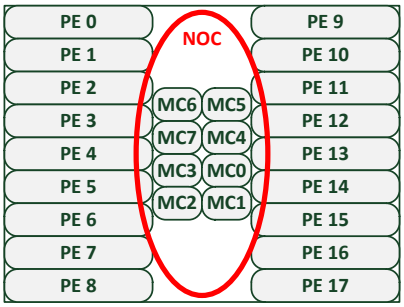

Fig. 4. On-chip topology of PEs and MCs

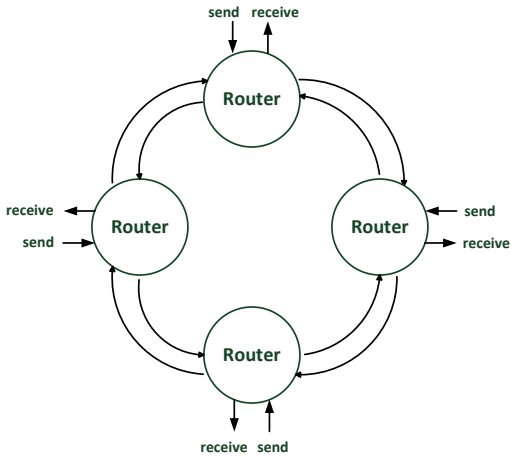

Fig. 5. Double-ring NoC topology

arbitration would control which PE would get access to that specific MC.

However, the placement constraints imposed by Convey in order to close 150 MHz timing tightly pack all MCIs into the center of the AE-FPGAs (see Figure 4). Attempting to route eight MARC back-end buses (each $>$ 128b wide) for *each* of the PEs leads to routing failure when more than five PEs are present in an AE, since there simply is not sufficient interconnect available in the center of the chip. As our work aims for more PEs per AE (at least a dozen should be supported), we needed a more scalable solution.

Thus, despite initial reservations with regard to the increased area and access latency, we also implemented a network-on-chip (NoC)-based communication architecture. Since our interest lies in prototyping parallel heterogeneous computing systems and not in NoC design, we were fortunate to be able to use the existing CONNECT NoC [17]. CONNECT presents a low barrier of entry, as the author makes not just a single NoC node router available, but provides a generator [16] as a web service for *entire* NoCs, which outputs synthesizable Verilog for the chosen configuration.

Considering the size of an individual r-VEX core (see Section VIII), it made sense to partition the V6LX760 chip

Bits	Description
31:0	ID for reordering and source port identification
95:32	Read / Write Data (64b)
159:96	Memory-address (64b)
167:160	Byte write Enable (8b)
168	read or write (1b)
169	virtual channel (at least 1b)
174:170	destination address (5b)
175	Tail (for multi-packets, unused in PHAT, 1b)
176	Valid (1b)

Table I. Structure of the NoC packets used in PHAT

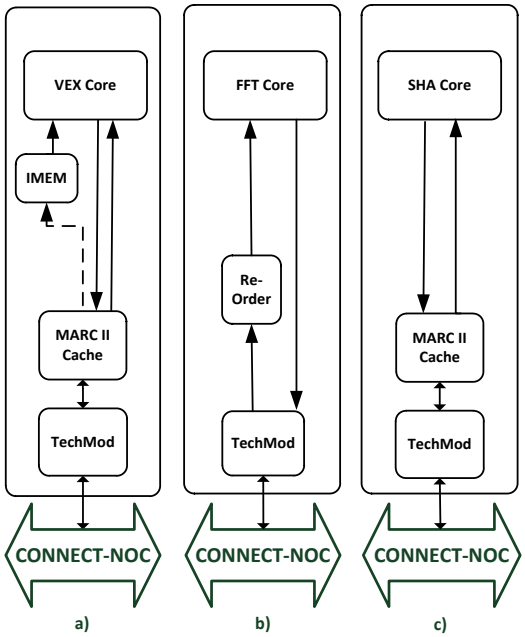

Fig. 6. Configurable interfaces between PEs and the NoC using PEMI

used for an AE into 18 regions around the central area holding the preplaced MCIs and movable NoC logic. Again with an eye toward routability, we performed floorplanning on these partitions (Figure 4) and chose a matching double ring topology (Figure 5, shows an example for four nodes) with a total of 26 endpoints in our largest configuration (18 PEs and 8 MCIs).

Table I shows the payload (169b) and control data (8b) of a PHAT NoC packet. Most of these data items also appear as the hardware signals to the CONNECT router blocks in the form of an easily-used parallel interface. As the number of Virtual Channels (VC) for the NoC instance is configurable at generation time, more channels (see Section VIII) will result in more bits being needed here.

VII. COMPLETE HARDWARE ARCHITECTURE

Using CONNECT, we provide each PE with a NoC router. Depending on the needs of the PE, an appropriate PEMI (Section V) is inserted between the router and the

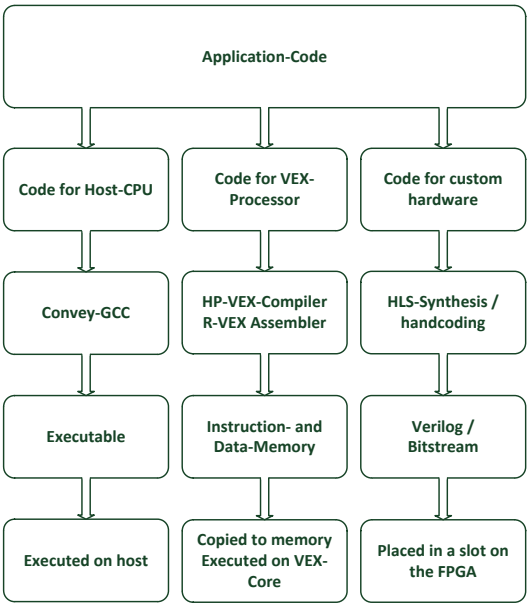

Fig. 7. Compiling for the PHAT prototype

PE itself: MARC II with separate IMEM for an r-VEX (Figure 6 a)), a plain Reorder Buffer for streaming accesses (Figure 6 b)), and MARC II for hardwired accelerators that do not need to fetch instructions (Figure 6 c)). A so-called Technology Module acts as the interface between the generic MARC/Reorder Buffer-internal protocols and the router interface.

This approach allows the easy integration of custom-compiled hardware functions generated by the Nymble [10] co-compiler, which already connect to the MARC II cache interface. Also, due to the simplicity of the protocol, it is easy to wrap manually designed IP blocks in the interface. Examples for both approaches will be shown in Section VIII).

VIII. EXPERIMENTAL EVALUATION

We evaluated the PHAT prototype using a number of software applications and two hardware functions, one automatically compiled using Nymble, the second one a manually wrapped IP core. For compilation to the different components of the prototype (x86 host CPU, r-VEX PE, accelerator PE), we used the three-pronged flow sketched in Figure 7), focusing here on compiling for the different PEs.

The r-VEX core is used in version 2.1, executables are compiled using version 3.43 of the VEX C compiler, assembled using TU Delft's r-ASM [19], Xilinx ISE 14.6 is used for logic synthesis, mapping, placement and routing.

VIII-A. Software Performance on r-VEX PEs

As described in Section III, we use 4-issue cores executing at 150 MHz (the default clock frequency of the Convey HC-1ex MC interface). Four software applications execute on the r-VEX PEs:

- **ADPCM**: Encoding of PCM values in shared memory, followed by decoding and comparison of the results.
- **Matrix Multiplication**: Multiplication of two dense 100x100 matrices in shared memory.
- **Bubblesort**: Sort of a 2000 element integer array in shared memory.
- **Contrast Enhancement**: Two-pass contrast-spreading on 256x256 8b greyscale images.

In summary, a single 4-issue r-VEX core clocked at 150 MHz has roughly 1/18 of the performance of one of the 2.13 GHz Xeon cores of x86 server host CPU. Performance for these sample applications scales linearly with the number of r-VEX cores, as MARC II, the NoC, and AEMS has sufficient bandwidth to never get saturated even when all 18 cores are operating on an AE. With four AEs available on the Convey HC-1ex, allowing a total implementation of 72 r-VEX cores, the prototyping performance of this PHAT platform is thus broadly equivalent to the native Quad-Core CPU, significantly exceeding the performance of HDL or instruction-set simulation.

VIII-B. Scalability of Homogeneous r-VEX Arrays

Table II gives an overview of the scalability of the approach. We show the required areas for the largest routable bus-based system (5x r-VEX), a smaller NOC based system with just eight r-VEX cores (leaving 10 partitions available for other logic), and the maximal NoC-based system with 18 r-VEX cores. For both NoC cases, we use CONNECT in its minimal configuration of just 2 VCs (one for sending, one for receiving of data). Sizes are given for an individual r-VEX core, a single MARC II cache (configured as described in Section V), an individual NoC router as well as the complete NoC. cae_pers is the size of the synthesized logic (PE array and NoC), while complete FPGA also includes the Convey-provided interfaces (e.g., the MCs).

In all cases, the complete FPGA met timing at 150 MHz. The bus-based implementation is of course somewhat smaller (ca. 10%) than the corresponding NoC-based one, but does not scale beyond the 5 PEs shown here. However, even for the largest PE arrays, the NoC requires just around 8% of the entire chip area.

VIII-C. Scalability of NoC Virtual Channels

If more network throughput is desired (e.g., for more cache-hostile applications) or when cache-coherency messages must also be transported (see Section IX), the bandwidth of the network can be scaled up by increasing the number of virtual channels. Table III shows the impact of scaling up to 4 and 8 VCs in a maximum r-VEX array. Even going to 8 VCs increases the total NoC just to 16%

	Slices	Slice Reg	LUTs	BRAMs
available on V6LX760	118560	948480	474240	720
5x r-VEX, bus-based				
complete FPGA	30521	45006	91144	132
cae_pers	19643	14815	62372	60
1x VEX-Core	2418	1401	8659	4
1x MARC II-Cache	833	409	1976	8
5x r-VEX, NoC w/ 2 VCs				
complete FPGA	33043	48915	104291	132
cae_pers	22712	18261	74241	60
1x VEX-Core	2418	1127	8511	4
1x MARC II-Cache	720	510	2071	8
CONNECT NoC	4520	2322	14123	0
1x CONNECT Router	320	169	1043	0
8x r-VEX, NoC w/ 2 VCs				
complete FPGA	45120	54915	141188	168
cae_pers	33693	24691	110311	96
1x VEX-Core	2418	1127	8511	4
1x MARC II-Cache	720	510	2071	8
CONNECT NoC	6226	3222	19258	0
1x CONNECT Router	324	171	1043	0
18x r-VEX, NoC w/ 2 VCs				
complete FPGA	81671	76843	260585	288
cae_pers	70450	46619	229134	216
1x VEX-Core	2632	1456	8781	4
1x MARC II-Cache	745	528	2100	8
CONNECT NoC	9626	5148	28747	0
1x CONNECT Router	370	198	1084	0

Table II. Area scaling of homogeneous r-VEX arrays

	Slices	Slice Reg	LUTs	BRAMs
available on V6LX760	118560	948480	474240	720
18x r-VEX, NoC w/ 4 VCs				
complete FPGA	84516	80027	265876	288
cae_pers	73188	49803	234572	216
1x VEX-Core	2710	1456	8781	4
1x MARC II-Cache	720	528	2111	8
CONNECT NoC	11270	8308	33851	0
1x CONNECT Router	430	318	1272	0
18x r-VEX, NoC w/ 8 VCs				
complete FPGA	91411	87883	282594	288
cae_pers	80568	57659	252012	216
1x VEX-Core	2710	1456	8781	4
1x MARC II-Cache	811	528	2105	8
CONNECT NoC	18763	16164	52294	0
1x CONNECT Router	720	618	1988	0

Table III. Area scaling of homogeneous r-VEX arrays with more VCs

of the total chip area, leaving sufficient scaling headroom to more traffic-intensive applications on the r-VEX cores.

VIII-D. Custom-Compiled Hardware PEs

To demonstrate the feasibility of quickly integrating custom-compiled hardware using PHAT, we used Nymble to compile the SHA kernel from the CHStone benchmark suite [3] into a PE for insertion into the PHAT array. Since the compiler has only limited optimization opportunities on the code (which uses unbounded loops and pointer accesses), the resulting hardware is significantly larger than the manually optimized r-VEX core. But it can be easily

	Slices	Slice Reg	LUTs	BRAMs
available on V6LX760	118560	948480	474240	720
15x r-VEX, 1x SHA, NoC w/ 2 VCs				
complete FPGA	88893	152505	286378	260
cae_pers	78822	122281	254911	188
1x VEX-Core	2487	1127	8497	4
1x MARC II for r-VEX	737	510	2101	8
1x MARC II for SHA	773	526	1941	8
1x SHA Accelerator	20121	81084	58548	0
CONNECT NoC	8404	14590	27651	0
1x CONNECT Router	325	171	1042	0

Table IV. Heterogeneous array combining r-VEX with compiled SHA accelerator

	Slices	Slice Reg	LUTs	BRAMs
available on V6LX760	118560	948480	474240	720
18x FFT, NoC w/ 2 VCs				
complete FPGA	63319	140492	165848	144
cae_pers	51942	110268	134713	72
1x FFT Core	2077	5047	5296	3
1x Reordering Buffer	89	303	186	1
CONNECT NoC	9537	5140	27880	0
1x CONNECT Router	371	171	1042	0

Table V. Homogeneous array of 18 FFT IP blocks

accommodated in PHAT just by allocating it to three of the 18 PE slots during floorplanning.

The heterogeneous array of 15 r-VEX cores and one SHA accelerator also meets 150 MHz timing on the Virtex 6 FPGA.

VIII-E. Manually-Optimized Hardware PEs

Both the homogeneous and the heterogeneous example arrays described above did not put large bandwidth demands on the memory system and the NoC. In order to stress-test these components, we now examine the integration of a manually optimized IP block that runs in full streaming mode.

To this end, we consider a pipelined 256 point FFT core [21], accepting a 32b input word and producing a 32b output word per cycle. At the system clock of 150 MHz, the total required memory bandwidth will be 1.14 GB/s per core. Since the core operates purely in streaming mode with no data reuse, it will not use a MARC II cache in the PEMI, but employ a simple Reorder Buffer (Section VII). As shown in Table V an instance of the FFT core (2077 slices) easily fits into one the r-VEX sized floorplan partitions (2800 slices), allowing us to execute the stress test with a homogeneous array of 18 FFT cores. Input and output is allocated as shared memory in the off-chip AEMS.

In this configuration of the on-chip MCIs (single channel mode), each AE has a maximal theoretical throughput of 9.15 GB/s to MCs in the AEMS. At best, however, we just reach 3.87 GB/s across all MCIs (shown for 9 FFT blocks active, Table VI). Using more VCs does not gain performance until we exceed five FFT blocks (no blocking

Number of FFT Blocks	2 VC	4 VC	8 VC
1	1.00	1.00	1.00
2	1.63	1.57	1.31
3	1.87	1.93	1.74
4	1.93	1.79	1.80
5	2.25	2.21	2.20
6	2.80	2.88	2.83
7	3.07	3.08	3.13
8	3.10	3.21	3.46
9	2.93	3.79	3.87
10	2.80	3.50	3.59
11	2.66	3.51	3.65
12	2.60	3.28	3.51
13	2.62	3.25	3.29
14	2.68	3.19	3.43
15	2.52	3.30	3.60
16	2.51	3.55	3.78
17	2.59	3.42	3.76
18	2.52	3.40	3.63

Table VI. Total throughput of the eight on-chip MCIs for FFT blocks [GB/s]

Number of FFT Blocks	2 VC	4 VC	8 VC
1	4.0	4.0	4.0
2	6.6	6.3	5.3
3	7.6	7.8	7.0
4	7.8	7.3	7.3
5	9.1	9.0	8.9
6	11.4	11.7	11.5
7	12.5	12.5	12.7
8	12.6	13.0	14.0
9	11.9	15.4	15.7
10	11.3	14.2	14.6
11	10.8	14.3	14.8
12	10.5	13.3	14.3
13	10.9	13.2	13.4
14	10.2	12.9	13.9
15	10.2	13.4	14.6
16	10.5	14.4	15.6
17	10.2	13.9	15.3
18	10.0	13.7	14.9

Table VII. Total throughput of the NoC for FFT blocks [GB/s]

occurs in the NoC). For six and more blocks, more VCs allow more packets to be in-flight between the PEMIs and MCIs. After exceeding 8 (for 2 VCs) or 12 blocks (for 4 or 8 VCs) the number of collisions has increased to a level that begins to reduce system performance again. So, it appears that the NoC is the bottleneck here.

We thus measured the throughput of the NoC itself: Reads induce two packets in the NoC (request and reply), while writes require only a single one. The results are shown in Table VII). To achieve a memory throughput of 3.87 GB/s for the FFT PEs, we have to process 15.7 GB/s in the NoC. So while the NoC did resolve the scaling issue the bus-based approach faces, there clearly is the need for more optimization of the NoC to satisfy even bandwidth-hungry applications.

IX. FUTURE WORK

While PHAT already allows the exploration and prototyping of parallel heterogeneous architectures using shared memory for communication much faster than software-based simulators (RTL or ISS), much potential for improvements remains.

One area of further development are the capabilities of the r-VEX core, which was originally never intended to be used in a chip-level multiprocessing (CMP) configuration. For more general use, key functions for parallel programming such as atomic update instructions will need to be added to the core.

Similarly, a major component missing from the current shared memory system is automatic cache coherence between PEs. Note that this capability is already supported in bus-based MARC II designs, but will need to be adapted to the NoC-based communication scheme. It should also be performed across AE boundaries, requiring inter-chip data routing between NoCs residing on different AEs.

Finally, to support even higher memory bandwidths, it is worthwhile to optimize the NoC itself. One approach would be the use of pipelining in the NoC, which is already present in CONNECT, but not reliable in all NoC configurations. The second would be the replacement of the CONNECT NOC routers with a faster version, e.g., as described in [9].

X. CONCLUSION

The PHAT prototype has already been useful in allowing experimentation with and evaluation of parallel heterogeneous architectures. For example, on 18 r-VEX cores, the contrast enhancement application described in Section VIII-A executes 1.4M times faster on PHAT than using software simulation.

The combination of selective floorplanning together with a scalable NoC architecture has permitted successful FPGA implementation even of complex PE arrays with short turnarounds, while still meeting the default timing of the underlying hardware platform.

The system is highly flexible, allowing customization and parameter tuning at many levels, from specific instruction set extensions of individual processors to altering global NoC characteristics.

Despite appearances, PHAT is also highly portable to different hardware platforms. While the currently employed Convey HC-1ex platform is desirable for its tight integration of a multi-core server-class x86 CPU with a large reconfigurable processing capacity, coupled to a high-throughput memory system, other use-cases might call for more economical solutions using smaller prototyping boards. In that case, only the MCI endpoint in PHAT would have to be altered, connecting the corresponding NoC node to the actual memory controller used. In smaller platforms, many more advanced features such as out-of-order memory

replies will most likely not even be required, thus avoiding the need for reorder buffers in the PEMI.

Work on alleviating some of the current limitations, specifically automatic cache coherence and higher available memory bandwidths, has already begun.

Acknowledgments: This work was supported by hardware and software donations from Xilinx Inc.

REFERENCES

[1] R. Avizienis, Y. Lee et al. RAMP Gold: A High-Throughput FPGA-Based Manycore Simulator. In *Design Automation Conference (DAC)*. 2010.

[2] D. Chiou, D. Sunwoo et al. FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators. In *MICRO*. 2007.

[3] CHStone - A Suite of Benchmark Programs for C-based High-Level Synthesis. [Online]. Available: http://www.ertl.jp/chstone/.

[4] E. S. Chung, M. K. Papamichael et al. ProtoFlex: Towards Scalable, FullSystem Multiprocessor Simulations Using FPGAs. 2009.

[5] J. Fisher, P. Faraboschi et al. *Embedded Computing*. Morgan Kaufmann, 2004.

[6] H. Gädke-Lütjens, B. Thielmann et al. A flexible compute and memory infrastructure for high-level language to hardware compilation. In *Field Programmable Logic and Applications (FPL), 2010 Intl. Conference on*, pp. 475–482. IEEE, 2010.

[7] P. Garcia and K. Compton. Kernel Sharing on Reconfigurable Multiprocessor Systems. In *Conference on Field Programmable Technology*. 2008.

[8] Hewlett-Packard Laboratories. VEX Toolchain. [Online]. Available: http://www.hpl.hp.com/downloads/vex/.

[9] Y. Huan and A. DeHon. FPGA optimized packet-switched NoC using split and merge primitives. In *Field-Programmable Technology (FPT), 2012 International Conference on*, pp. 47–52. 2012.

[10] J. Huthmann, B. Liebig et al. Hardware/software co-compilation with the Nymble system. In *2013 8th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, pp. 1–8. IEEE, 2013.

[11] R. Kumar, D. M. Tullsen et al. Heterogeneous Chip Multiprocessors. *Computer*, 38(11):32–38, 2005.

[12] R. Kumar, D. M. Tullsen et al. Core Architecture Optimization for Heterogeneous Chip Multiprocessors. In *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*. 2006.

[13] H. Lange and A. Koch. An execution model for hardware/software compilation and its system-level realization. In *Field Programmable Logic and Applications, 2007. FPL 2007. Intl. Conference on*, pp. 285–292. IEEE, 2007.

[14] H. Lange and A. Koch. Architectures and Execution Models for Hardware/Software Compilation and Their System-Level Realization. *IEEE Trans. Comput.*, 59(10):1363–1377, 2010.

[15] H. Lange, T. Wink et al. MARC II: A Parametrized Speculative Multi-Ported Memory Subsystem for Reconfigurable Computers. In *DATE*. 2011.

[16] M. Papamichael. CONNECT-Generator. [Online]. Available: http://users.ece.cmu.edu/ mpapamic/connect/.

[17] M. K. Papamichael and J. C. Hoe. CONNECT: Re-Examining Conventional Wisdom for Designing NoCs in the Context of FPGAs. In *FPGA*. 2012.

[18] M. Pellauer, M. Adler et al. HAsim: FPGA-based high-detail multicore simulation using time-division multiplexing. 2011.

[19] r-VEX: Sources and Tools. [Online]. Available: https://code.google.com/p/r-vex/.

[20] R. Seedorf, F. Anjam et al. Design of a Pipelined and Parameterized VLIW Processor: r-VEX v.2. In *Proc. 6th HiPEAC Workshop on Reconfigurable Computing*. Paris, France, 2012.

[21] Uzenkov, Oleg and Sergiyenko, Anatolij . Pipelined FFT/IFFT 256 points processor [Online]. Available: http://opencores.org/project,pipelined_fft_256.