A Universal Processor for RAMP

Greg Gibeling and John Wawrzynek
{gdgib and johnw}@eecs.berkeley.edu

June 21, 2006

1 Introduction

The objective of the RAMP project is to accelerate multiprocessor systems research through the emulation of usable parallel processor prototypes. Criteria such as low cost, flexibility, observability, credible and repeatable results, and reasonable performance led to the choice of the FPGA platform, BEE2 (Berkeley Emulation Engine) as the primary host environment.

A key goal of RAMP is the extraction of credible performance results, even from tests which are not run in real time. The need for a timing model was the original reason for RDL [12].

Credible results, however, also require the use of known tests, or applications, many of which are tied to existing instruction set architectures (ISAs). To run such tests, and finish the infrastructure phases of RAMP quickly, it is highly desireable to be able to run pre-compiled code for existing architectures. This has led to a major push among the RAMP participants to find, adapt or develop HDL descriptions of existing ISAs and, with the addition of RDL wrappers, turn them into RAMP-compatible processors. While this approach will lend signficant credibility to any results, it presents critical challenges, and will delay the project from the longer-term goal of parallel system research. In this paper we outline our proposal to drastically reduce the one-time work required to achieve ISA level compatiblility for a variety of architectures, while addressing essential research and efficiency goals.

Rather than trying to acquire and adapt for FPGAs, HDL code for each processor family and model, we propose to build a single unified processor, along with tools to customize and program it. When compatibility is required, exiting code can be made to run on this processor by using binary translation. Because we would no longer attempt to synthesize gateware from an existing processor simulation model, *e.g.*from OpenSPARC, our unified processor can be dramatically more area-efficient, as it can have an ISA and micro-architecture tailored to FPGAs. Furthermore, by generating (possibly widely varied) processors in a uniform way, we can significantly simplify research into architectural features.

The next few sections cover the problems, goals and research questions this project will address. As this is a preliminary whitepaper, we do not offer complete solutions but we do attempt to suggest them for most of the open research questions. We are also openly soliciting related ideas as well as informed suggestions and commentary.

2 The Current Approach

The current RAMP approach is typified by the RAMP Blue project, where multiple instances of an FPGA implementation of a commercial processor core, the Microblaze from Xilinx in this case, are combined to form a simple multi-core design. The MicroBlaze was chosen as the first RAMP soft processor, as it had an existing efficient mapping to FPGAs. From its conception, the MicroBlaze ISA and microarchitecture were optimized for FPGA implementation, and its low-level mapping to FPGAs was hand-optmized.

By providing a quick path to a compact multi-core system, The MicroBlaze nicely served the short term needs of RAMP. However, in the long-run, RAMP will need to support several more commercially relavent ISAs, such as PowerPC, SPARC, ARM, X86, and MIPS. Unfortunately, these ISAs have not been optimized for FPGA implementations, nor have their currently available microarchitectures.

Furthermore, none of these cores have optimized low-level FPGA implementations. While in some cases, we can expect to rely on processor vendors to do the work necessary to create efficient FPGA versions of their processor cores, adapting these cores for RAMP would require significant investment by the RAMP team to get these cores to a point where they are practical for RAMP. Evidence of this is visible in a simple comparison of LUT counts: a full, unoptimized OpenSPARC core would take 130,000 LUTs on a Xilinx Virtex4, whereas a Xilinx MicroBlaze can take less than 2000. With the current generation of boards, the BEE2, this is the difference between having 2 and 40 processors per board.

In the next section, we present our solution to these problems.

3 Vision and Proposal

We propose to create a set of tools and languages to allow a researcher to specify, build, compile for, debug, and run existing code on new and possibly specialized processor architectures. Setting aside the challenges in the creation of such a toolset, which we will discuss in 4, we believe this presents a clean solution to the problems outlined in 2

The vision for RAMP centers around the idea that it could become the shared platform for multiprocessor architecture and systems research, in such a way as to supplant all others. Ideally, this would include a simple set of tools, perhaps with a GUI, to create a usable computer system, including processors, memory, network, storage and I/O, which has been tuned for efficient implementation and can run existing applications.

Such a system could include the flexibility needed to make architectural changes, *e.g.*ATLAS [20] transactional memory support. It could also include flexibility in the network design, allowing a researcher to experiement with the performance of various topologies. Because such a system would naturally require specification at a very high level, efficient implementations could be generated, increasing the number of procecssors, while decreasing the slowdown relative to a full custom ASIC design. Furthermore, with the ability to run precompiled binaries, modifications could be made independant of operating system and application compilation, drasticly reducing the time to do detailed performance studies.

We believe that even the processor architecture subset of such a toolkit would meet most of the infrastructure goals of the RAMP project, leaving participants free to pursue new research rather than recoding old ISAs for FPGA implementation. The efficiency gains from such a universal processor compiler targetted to an FPGA would provide the cost reduction needed to run high performance processors. The flexibility and code compatibility would allow the use of existing code, even to the point of running closed source operating systems such as Microsoft Windows. We should also mention that there are existing projects are already looking at the network generator portion of the overall system toolkit.

The primary features of our proposed toolkit are the architecture compiler, and binary translation framework. By synthesizing processor cores from high level descriptions, including both the ISA, and overall architecture (number of pipeline stages, cache size, etc), we can more easily and efficiently target FPGAs. Tricks like multiplexor implementation through registers, the use of double clocked Xilinx Block RAM for dual port register files, and so forth can be relatively painlessly added to such a generator, especially as part of the larger RDLC3 and RCF (RAMP Compiler Framework) implementations. Furthermore, given a semantically rich description of an ISA, including ISA, consistency model, virtual memory and exceptions, we should be able to automatically generate binary translators, allowing easy customization of the underlying processor with the need to even re-compile applications.

In this section we have outlined our ideal full computer system generator tools, and describe the subset, a universal processor compiler and binary translator generator, which will accomplish most of our goals. We have briefly mentioned their integration with the next generation of RDLC and RAMP compiler tools through the generalized RCF. However, we have not addressed the myriad of challenges to the creation of these tools, nor provided any substantial justification for our belief that their creation is possible; we will do that in the following section 4.

4 Challenges

In the previous section we outlined high level goals and long term vision, without regard to the challenges inherent in our proposal, in this section we outline and begin to address these issues. First of all, the universal processor generater will entail a complete micro-processor synthesis tool, which should include not only specialized instruction synthesis, but generators for various microarchitectures, from single cycle to out of order. Furthermore, efficiency issues of FPGA implementations remains a daunting task in the face of this kind of generator framework. Second, binary translation has been an active area of research for some 20 years, spanning many universities and companies, but there have been relatively few large scale successes.

4.1 Universal Processor

While there have been a number of projects, and even a commercial product from Tensilica, which generate application specific processors, what we are proposing is an order of magnitude more ambitious. Clearly the creation of a functional datapath from an NJMC [24, 25], SSL [5] or application level description [19, 14, 28] is a relatively simple problem, as commercial synthesis tools general can create reasonable datapaths. However, the creation of efficient processor execution stages adds the complication of control organization, which in the case of a microarchitecturally parameterized generator, such as the one we propose, will require new research. Furthermore, most of the existing projects deal in fixed memory consistency, virtual memory and exception models, all of which we would like to parameterize or describe in an application specific language.

The primary isssues facing the universal processor generator work are:

1. Memory coherence and consistency models for cross processor compatibility.
2. Highly compact ISA synthesis in FPGAs.
3. Abstract virtual memory models for compatibility.
4. Full instruction set virtualization.
5. Support for binary translation, both as a target and host.

Our justification for building a generator rather than a single processor implementation stems from the similarity of processors, the need to trade area for speed in research experiments, and the tantalizing opportunity to build integrated tools using the core RDLC3 and RCF code to tie together comilers, languages and debugging tools.

The primary importance of the area-speed tradeoff arises from the desire to build 1000 processor machines and still perform credible timing simulations. These two goals will often be in conflict, and a processor generator will allow researchers to trade the two, perhaps using coarser, faster simulations early in a project to guide further work, a lofty goal which is painfully lacking in our own current work.

The desire to build integrated tools for such a processor environment is less clear, but stems from our interest in building drastically different ASIPs. We envision being able to accomodate everything from a full out of order processor, to a 20 LUT bit serial proccessor such as might be useful for system initialization tasks. Having a common set of tools for this wide range of implementations will allow us to save significant work, especially in the complex FPGA specific optimization areas, and compiler building, which RCF is specifically designed to simplify.

We believe that a universal processor generator is both possible, given the state of research and our FPGA targets, and useful given the time it would take to mindlessly implement a range of existing ISAs. Our hope is that this will both reduce the time to build a complete RAMP infrastructure, and provide ways of doing new research.

We would like to point out that over the past two months there have two projects involving one of the authors building specialized processors. The FLEET builder project comprises a series of RDLC2 plugins to build a customized processor (based on a special architecture [26]) and the appropriate assembler from a very simple listing of functional units. Similarly, there was a course project [13] to build an implementation of the P2 [21] system on top of RDL, which included a special purpose database tuple processor builder, along with a customized assembler and compiler for this processor. While these projects are far simpler than the universal processor generator, they provided confidence that we can build such a tool on the RDLC3 and RCF framework.

4.2 Binary Translation

The use of binary translation to run precompiled code, is the linchpin of our argument for using the Universal Processor as the core of RAMP work. There are several arguments against binary translation, starting with correctness and performance.

Most past projects, UQBT [2, 3, 7, 4, 5, 6] excepted, have focused solely on translation from fixed ISAs, most of which have closely matched the host and target architectures. Of course our proposal includes translation between vastly different architectures, incurring both performance problems for emulation of mismatched instructions and correctness issues especially in virtual memory, consistency and exceptions. This again is a key motivation for the creation of a universal processor generator: projects like Embra [29] and BOA [1, 11, 15] suggest that even relatively minor changes to the host ISA can have significant positive impact on the speed of translated binaries. Because we are primarily interested in FPGA implementations (we are not currently investigating ASICs, but we are not ruling them out) our host processors can be flexible enough, that should an experiment require better performance a researcher can simply tweak the host ISA.

The fact is that there have been a wide range of binary translation projects spanning over two decades of research, including UQBT (which became Sun Walkabout [8]), Embra, BOA, DAISY [16], FX!32 [10], the Transmeta Crusoe [9] and the

Intel Pentium4 [18]. Because there exists a large, but somtimes incomplete, body of knowledge, we believe there is a firm basis for our work. UQBT in conjunction with NJMC suggest ways of describing, if not specifying processors such that translation tools can be automatically generated. BOA and Crusoe both provide results which can guide our choice of translation time (static, dynamic or instruction level). The FX!32 and Pentium 4 both provide successful examples of such software and hardware under deployment conditions. And papers like [16] provide extensive arguments for the use of the techniques proposed in this whitepaper.

The most compelling argument against binary translation is the overhead introduced into running even the simplest of applications. Systems like Embra, however, brought this down to a 4x slowdown for the fastest case, from a more common 20x-100x in previous systems. We believe that between the clever techniques used by such systems, and the possible customization of the host processor in extreme cases, the performance of our translations will be acceptable. In addition, we intend to stress the use of run-time optimizations, which in some cases have caused binary translation to accelerate unprofiled applications. In light of the customizability of the host processor, poor performance is more likely to be a result of a ISA gap between the host and targets, which can be reduced by customization.

However, the real key performance observation is that raw speed has NEVER been a goal of the RAMP project. By trading some performance for observability, reproducability and flexbility we can build a much better research platform. In the end, it is researcher time, the late hours put in by graduate students, professors and even undergrads, which is the most costly. We believe that our universal processor and binary translation tools with successfully trade modest amounts of run time for enormous amounts of mindless researcher work time, a very beneficial trade indeed, and the basis of all computing.

4.3 Compatibility and SMC

Some of the biggest challenges in binary translation arise from the uncommon cases: exceptions and self-modifying code are the two biggest challenges in most of the binary translation research. In general they are handled the same way: by invocation of dynamic translations or emulation, trading speed for power and flexibility in the uncommon case.

Combinations of the work on Embra and the Crusoe also suggest interesting possbilities for debug and test instrumentation, and optimization. First off, both Embra and UQDBT [27] (the dynamic variant of UQBT) allow the instrumentation of the translation results to provide performance counters accurate to the target architecture. Similar techniques could be used to allow *e.g.*IA-32 (x86) cache studies even under a universal processor and binary translation scheme.

Second, the Crusoe work makes a critical distinction between truly self-modifying code, the hardest case for translators, and seemingly self-modifying code, wherein data and code are on the same VM page. Some of the techniques in [9] suggest efficient ways to differentiate and handle these cases, thereby avoiding some of the most common drawbacks to binary translation.

4.4 Multiprocessor Systems

To this point we have repeatedly mentioned the need for small and efficient generated processor in order to build large multiprocessors, but we have not addressed *e.g.*the challenges of binary translation on these systems. Clearly the univeral processor generator must handle multiprocessor synchronization in a predictable manner. It will need to include support for everything from networked or shared nothing, to cache coherent systems. Work on such project as the CRF [30, 22] memory model, perhaps with added support for transactions such as those under investigation [17, 20, 23] or those used by the Crusoe, should provide a basis for these various synchronization systems and translation between them.

Because even RAMP Blue exhibits under utilization of some system level resources, in this case memory bandwidth, we see opportunities for sharing, especially among processors on one FPGA. For example, with a translation cache, translation of code is a relatively infrequent operation, meaning that many processors might share a single hardware translator, or processor dedicated to translation. Furthermore, as most software scenarios for RAMP machines call for all processors to run a single OS, most translations could be shared by multiple cores, thereby reducing the high memory requirements as reported by Embra.

5 Conclusion

There exists a large body of research in binary translation, and yet opportunities like an FPGA implementation, and challenges like a large multicore system present opportunities for new research. There have also been many projects aimed to create application specific, or parameterized processors and descriptions and yet the relatively performance insensitive application of RAMP, coupled with it's research nature offers a new opportunity to use these tools.

In conjunction we believe that a universal processor generator and binary translator toolset will provide an acceptably high performance, platform for the research goals of the RAMP project. Furthermore, the creation of these tools, followed by their use will replace the time consuming, mindless implementation of existing ISAs with important research into new processor design and specification techniques. This project represents not only an opportunity to further RAMP, but an interesting application of the RAMP project to previous research on ISA design and binary translation.

As a final note, while we have dedicated signficant research to the opinions in this paper, they remain opinions. The fact is that there will be little proof either for or, and we stress this point, against the ideas presented here without significantly more research, and we are currently open to both ideas and opinions, even as we are commited to moving forward. We would also like to note that this research does not preclude the usefulness of implementing some ISAs directly, for validation or comparison, if not for the clearly higher efficiency.

We would like to thank Alex Krasnov and Adam Megacz for their help shaping the ideas presented in this paper, and their ability to point out our obvious mistakes.

References

[1] E. Altman, M. Gschwind, S. Sathaye, S. Kosonocky, A. Bright, J. Fritts, P. Ledak, D. Appenzeller, C. Agricola, and Z. Filan. Boa: The architecture of a binary translation processor, 1999.

[2] C. Cifuentes. Partial automation of an integrated reverse engineering environment of binary code. In L. Wills, I. Baxter, and E. Chikofsky, editors, *Proceedings of WCRE '96: 4rd Working Conference on Reverse Engineering. Monterey, CA*, 1996.

[3] C. Cifuentes and M. Van Emmerik. Recovery of jump table case statements from binary code, 1998.

[4] C. Cifuentes and V. Malhotra. Binary translation: static, dynamic, retargetable? In *Proceedings of International Conference on Software Maintenance. Monterey, CA*, 1996.

[5] C. Cifuentes and S. Sendall. Specifying the semantics of machine instructions. In *Proceedings 6th International Workshop on Program Comprehension. IWPC'98. Ischia, Italy. IEEE Comput. Soc. Tech. Council on Software Eng. 24-26 June 1998*, 1998.

[6] C. Cifuentes and Doug Simon. Procedural abstraction recovery from binary code, 1999.

[7] Cristina Cifuentes, Mike Van Emmerik, Norman Ramsey, and Brian Lewis. Experience in the design, implementation and use of a retargetable static binary translation framework, 2002.

[8] Cristina Cifuentes, Brian Lewis, and David Ung. Walkabout a retargetable dynamic binary translation framework.

[9] J. C. Dehnert, B. K. Grant, J. P. Banning, R. Johnson, T. Kistler, A. Klaiber, and J. Mattson. The transmeta code morphing software: using speculation, recovery, and adaptive retranslation to address real-life challenges. In *International Symposium on Code Generation and Optimization. CGO 2003. San Francisco, CA*, 2003.

[10] Paul J. Drongowski, David Hunter, Morteza Fayyazi, and David Kaeli. Studying the performance of the fx!32 binary translation system.

[11] K. Ebcioglu, J. Fritts, S. Kosonocky, M. Gschwind, E. Altman, K. Kailas, and T. Bright. An eight-issue tree-vliw processor for dynamic binary translation. In *Proceedings International Conference on Computer Design. VLSI in Computers and Processors. Austin, TX*, 1998.

[12] Greg Gibeling, Andrew Schultz, and Krste Asanovic. Ramp architecture and description language, 2005.

[13] Greg Gibeling, Andrew Schultz, and Nathan Burkhart. Combining p2 and rdl to build dataflow hardware programs, 2006.

[14] M. Gschwind. Instruction set selection for asip design. In *Proceedings of the International Conference on Hardware and Software. Rome, Italy. ACM SIGDA. IEEE Comput. Soc.. ACM SOGSOFT. IFIP WG 10.5. 3-5 May 1999*, 1999.

[15] M. Gschwind and E. Altman. Precise exception semantics in dynamic compilation. In

R. N. Horspool, editor, *Compiler Construction. 11th International Conference, CC 2002 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002. Proceedings. Grenoble, France. 8-12 April 2002*, 2002.

[16] Michael Gschwind, Kemal Ebcioglu, Erik Altman, and Sumedh Sathaye. Daisy/390: Full system binary translation of ibm system/390, 1999.

[17] L. Hammond, B. D. Carlstrom, V. Wong, B. Hertzberg, M. Chen, C. Kozyrakis, and K. Olukotun. Programming with transactional coherence and consistency (tcc). In *Eleventh International Conference on Architectural Support for Programming Languages and Operating Systems. Boston, MA*, 2004.

[18] Glenn Hinton, Dave Sager, Mike Upton, Darrell Boggs, Doug Carmean, Alan Kyker, and Patrice Roussel. The microarchitecture of the pentium 4 processor. *Intel Technology Journal*, 2001.

[19] Huang Ing-Jer and A. M. Despain. Synthesis of instruction sets for pipelined microprocessors. In *Proceedings of 31st ACM/IEE Design Automation Conference. San Diego, CA*, 1994.

[20] Christos Kozyrakis and Kunle Olukotun. Atlas: A scalable emulator for transactional parallel systems, 2005.

[21] Boon Thau Loo, Petros Maniatis, Tyson Condie, Timothy Roscoe, Joseph M. Hellerstein, and Ion Stoica. Implementing declarative overlays, 2005.

[22] J. W. Maessen, Arvind, and Shen Xiaowei. Improving the java memory model using crf. In *OOPSLA 2000. Conference on Object-Oriented Programming Systems, Languages and Applications. Minneapolis, MN*, 2000.

[23] Kevin E. Moore, Jayaram Bobba, Michelle J. Moravan, Mark D. Hill, and David A. Wood. Logtm log-based transactional memory. In *International Symposium on High Performance Computer Architecture (HPCA)*, pages 1–12, 2006.

[24] N. Ramsey and M. F. Fernandez. The new jersey machine-code toolkit. In *Proceedings USENIX Winter 1995 Technical Conference. New Orleans, LA*, 1995.

[25] N. Ramsey and M. F. Fernandez. Specifying representations of machine instructions. *ACM Transactions on Programming Languages and Systems*, 19(3):492–524, 1997. Publisher: ACM, USA.

[26] Ivan Sutherland. Fleet a one-instruction computer, August 25, 2005 2005.

[27] D. Ung and C. Cifuentes. Machine-adaptable dynamic binary translation. In *ACM SIGPLAN Workshop on Dynamic and Adaptive Compilation and Optimization (Dynamo'00). Boston, MA*, 2000.

[28] J. Van Praet, G. Goossens, D. Lanneer, and H. De Man. Instruction set definition and instruction selection for asips. In *Proceedings of 7th International Symposium on High-Level Synthesis . Niagara-on-the-Lake, Ont., Canada. IEEE Tech. Committee on Data Autom.. ACM SIGDA. 18-20 May 1994*, 1994.

[29] E. Witchel and M. Rosenblum. Embra: fast and flexible machine simulation. In *1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems. Philadelphia, PA*, 1996.

[30] Shen Xiaowei, Arvind, and L. Rudolph. Commit-reconcile and fences (crf): a new memory model for architects and compiler writers. In *Proceedings of the International Symposium on Computer Architecture. Atlanta, GA*, 1999.