Compilation for Future Nanocomputer Architectures

Thomas P. Way
Applied Computing Technology Laboratory
Department of Computing Sciences
Villanova University, Villanova, PA 19085
thomas.way@villanova.edu

Abstract

Compilation has a long history of translating a programmer's human-readable code into machine instructions designed to make good use of a specific target computer. In this paper, we formalize a compiler framework that broadly defines the task of compilation to include output of a machine description customized to the input program which would be used to generate the target computer. The compiled program would then run on the generated computer. Inspired by research in design space exploration, this compilation approach exploits the proposed capabilities of nanocomputers, which are in the class of reconfigurable parallel architectures. This emerging hardware technology relies on molecular level fabricated circuit design to minimize feature size while creating a vast matrix of reconfigurable processing units, an application of the advancing field of nanotechnology. We identify design issues and present preliminary results that support earlier work in this area and propose future directions.

Keywords: Nanocompilers, nanocomputers, high performance computing, reconfigurable computing.

1 Introduction

Traditional compilation for high performance computers relies on a wide variety of sophisticated analyses and optimizations to translate a source language program into efficient binary machine code for a specific target architecture [1]. To perform this translation, the compiler parses the source program, applying machine-independent optimizations on an abstract intermediate form of the program. Next, machine-dependent optimizations are performed with the goal of producing hopefully a significant improvement to performance of the resulting executable program. These improvements are made with the specific attributes of the target machine, or processor, on which the program will run in mind. [1]

Different processors can differ dramatically in the code improvements which work best for them. For instance, a program compiled to run optimally on one specific architecture may perform poorly on a different architecture. As a result, at least a portion of the compiler, the back-end, must be retargeted, potentially for each different processor or class of processors. [7] Although machine-independent optimizations can produce significant improvements in general [1], machine-specific knowledge, such as knowing the number of registers, memory organization and cache structure, instruction set, functional units available, and any other explicit parallelism that may be available, as with EPIC (Explicitly Parallel Instruction Computing), Superscalar and VLIW (Very Long Instruction Word) processors, is required to achieve significant results on modern, high-performance architectures. [1,14]

If Moore's Law is to continue to be applicable as a predictor [18], significant advances in chip manufacturing techniques are needed. In modern processors, feature sizes have been reduced while heat dissipation needs have increased to the point where there is a growing consensus that Moore's Law may actually be reaching an insurmountable barrier dictated by physics this time [12]. Nanotechnology, manufacturing performed through manipulation of atoms and molecules [8], is capable of overcoming this most recent barrier.

Nanocomputer architectures, produced using this molecular manufacturing approach, provide a natural successor to current general-purpose microprocessor architectures [9]. Nanocomputers, of course, must be functionally at least as capable as their predecessors, fast, inexpensive, robust and capable of operating at room temperature and of executing legacy code [2]. Whether Moore's Law truly loses applicability, or we merely readjust our thinking about it, it is clear that there are significant hurdles to overcome in computer design in the very near future.

Nanocomputing is discussed in this paper within the context of the topic of reconfigurable computing, which includes Field Programmable Gate Arrays (FPGA), Field-programmable Custom Computing Machines (FCCM), cellular array architectures, and synthetic neural systems, among many others [2,7,9]. The attractive capabilities of these emerging technologies include the ability to dynamically reconfigure or redesign the functionality of the processor, to produce nanoscale features (transistors, gates, logic circuits) that are at the very limits of physics

leading to processors which are many times smaller, more powerful and more efficient that those possible through current manufacturing techniques, and to generate a new, customized processor as simply as we now generate a new executable program [2].

In this paper we propose a compilation framework that could be used to produce a processor based on a source code program such that this new processor would be ideally suited to running the compiled program. Much as traditional compilers customize the program to suit the machine, we propose a compiler that customizes the machine to fit the program. Our research is on the demand side of nanocomputing rather than the supply side; although no such reconfigurable nanocomputer architecture of such a grand scale yet exists, trends point to the development of molecular- and atomic-scale switches in the near future [6]. We present the results of preliminary experiments designed to confirm earlier results in design space exploration [25] within a high performance EPIC machine, and extrapolate these results to the increasingly realizable promise of nanotechnology [8].

2 Background & Motivation

With limitations of miniaturization within chip manufacturing approaching the bounds of what is physically possible with known techniques, new methods of manufacturing are needed. Although some success has been achieved with extremely specialized multiprocessor systems using current technology, such as IBM's "Blue Gene" project [15], such large scale multiprocessor systems suffer from significant interprocessor bottlenecks. Development of scalable interprocessor communication schemes for million or billion processor systems has been slow [9], although it is viewed as a fundamental challenge for the future of processor design [11,17,21]. A wavefront approach to reconfiguration and interprocessor communication for such large scale processors has been proposed as a feasible approach [9].

Nanotechnology provides promise as a manufacturing approach that can address these current physical limitations, and indeed research in applications of nanotechnology to logic gate design and chip manufacturing is active and well-funded in the United States, Japan and elsewhere [20]. There is little doubt that nanocomputers, consisting perhaps of millions or billions of reconfigurable processing cells, laid out in a vast, reconfigurable fabric, are technically feasible and likely to be realized [2,8,9]. Our work attempts to design a compilation approach that will exploit the capabilities of these highly flexible future machines.

2.1 Nanotechnology

The field of nanotechnology is a promising approach to manufacturing through the direct manipulation of atoms and molecules, accomplished through some combination of chemical, electrical and physical interactions [8]. Although the hype surrounding the great promise of this potentially revolutionary technique is vast, reminiscent of the ballyhoo in the 1970s regarding the coming age of artificial intelligence (a robot in every home to do one's bidding), it is undeniable that it is a viable manufacturing approach that is gaining considerable momentum [20].

Nanotechnology consists of a group of technologies, just emerging, which enable matter to be manipulated at the nanometer scale. This infinitesimal scale, on the order of a small number of atoms, is at the lower limits of physics as they are now understood, thus their applicability to solving current feature-size hurdles of chip manufacture, for example. Less precise applications of nanotechnology are already producing useful products in the pharmaceuticals industry, and the techniques are emerging that will allow more sophisticated products to be produced using much more precise control of matter.

Although there is controversy surrounding the future benefits of nanotechnology, current trends point to the development of artificial molecular machine systems capable of building complex systems to atomic precision. In effect, we will duplicate the ability of, for example, a cow to convert hay into beef, a feat of molecular engineering that is currently performed quite effectively by the cow. The result of this bottom-up approach to manufacturing could be products that cure cancer by repairing cells, replace fossil fuels with clean and efficient alternatives, change the color of paint on a wall with the touch of a finger, and generate massively parallel and reconfigurable computers the size of a thumbtack. Microwave oven sized advanced molecular manufacturing systems will be capable of making large, useful products, very cheaply and with tremendous precision. Acceptance of these provocative predictions is contingent, of course, upon demonstrable results. [8,10,16]

Along with the predictions of amazing innovations come those of enormous danger. An example of such a nanotechnological nightmare is the "gray goo" problem, where a self-replicating nanomachine, and its exponentially increasing progeny, run amuck, breaking down all of the matter encountered into a uniformly gooey gray substance. The objective, then, is to proceed cautiously with the realization that a "bug" in a nanotechnology program could

potentially cause more damage than one that now leads to a seemingly quaint Blue Screen of Death. Exhaustive research is critical to the establishment of safe approaches to the development and use of nanotechnology. [8]

2.2 Reconfigurable Architectures

As mentioned before, a wide variety of successful hardware approaches exist for reconfigurable computing [9]. As physical limitations of feature size reduction and heat dissipation are reached, nanotechnology provides an approach to overcome these limitation. Because nanotechnology could lead to inexpensive production of highly reconfigurable computer hardware, it is natural to extrapolate the current state of the art to this emerging technology. Research strongly suggests that reconfigurable architectures, if efficient, will provide a better fit and thus improved performance for general purpose computation. [2,4,7,9]

Recent advances in design space exploration as applied to compilation for FPGA-based and other reconfigurable architectures demonstrates the performance improvements possible with customized architectures [25-27]. The concept of "program in, chip out" (PICO) relies on compiler analysis, particularly targeted automatic parallelism, to identify program fragments that will benefit most from customized hardware. [27]

Future nanocomputer architectures will be formed from non-volatile reconfigurable, locally-connected hardware meshes that merge processing and memory. This architecture models more closely than any existing computer architecture that of the human brain, with its vastly superior computing power [19]. Although the nanocomputer does not yet exist, the performance of any program can benefit from reconfiguration of the machine on which it runs to better match the resource requirements of the program. In contrast to an approach using design space exploration, the potential flexibility of nanocomputer architectures means that configuration will not be limited to a known design space.

2.3 Architectural Features & Compiler Analysis Techniques

Certain characteristics are common to most processors, and should be considered as candidates for reconfiguration. Characteristics such as the number of registers, functional units, memory units, cache and memory organization, branch history and prediction support, pipeline organization, and the instruction set architecture (ISA) should be considered. [14] Compilers currently retarget a source program to a target machine. With the flexibility of machine reconfiguration, the task of retargeting is made more powerful because the resulting target machine will be better suited to the specific characteristics and requirements of the source program.

In analyzing the characteristics inherent in a given source program, the ability to detect instruction-level, task-level and coarse-grained parallelism will also guide the description of architectural features. For instance, the amount of instruction-level parallelism (ILP) available in a program can impact the number of functional units that are specified, leading to improved performance. Rather than computer architects investing significant money and time in the design of an excellent general-purpose processor that does reasonably well running most source programs, the compiler itself can generate a well-tuned machine configured to match the individual source program, allowing each to maximize performance.

The compiler has the responsibility of analyzing the source program and generating a suitable machine description based on the results of the analysis. There are extensive analyses available, although the majority is designed to optimize the program for a specific target machine [1]. Any analyses that measure or enable the measurement of variable liveness, inherent parallelism, branching behavior, common subexpressions, memory access patterns, cache utilization, register allocation opportunities, instruction scheduling opportunities, and many other characteristic-related metrics are candidates. All can provide information to guide the generation of a machine description that is suited to the source program. The key consideration is the ability to gather information about static characteristics or predictive behavior of the source to enable production of a best-fit machine description.

3 Compiler Design

The major goal of the initial work reported in this paper is the design of a compiler that generates as its output both an executable version of the original source program and the description of a machine on which the executable will run well. This study focuses on ILP architectures, although the concepts also apply more broadly to include coarse-grain approaches. Traditional compilers take the source code and translate it into a binary form suitable for a specific processor, optimized to run as well as possible on that target machine. Knowledge of the target machine is needed to perform machine-dependent optimizations. Our approach differs in that the configuration of the target

machine is unknown when compilation begins. The machine configuration is extracted from the source program, based on its resource requirements. In this way, the resulting machine is an excellent fit to the program.

To discover this machine description, the compiler must perform a number of analyses designed to reveal the needs of the program. For example, live variable analysis can be performed to help determine how many registers are needed. However, the number of registers needed also depends on the issue width of the target machine, which is determined by the inherent ILP in the program. Thus, there are numerous interactions to consider when analyzing a program and generating a machine description.

Figure 1 illustrates the organization of the proposed nanocompiler from a high level. Source code is processed by the **Front end** of the compiler, including machine-independent optimizations. The resulting intermediate form is passed to a **Machine requirements analysis** phase, which performs static analysis, providing metrics to the **Machine description generation** phase. The resulting machine description is used by the **Processor generator** phase to generate or reconfigure the target machine, and by the compiler **Back end** to perform machine-dependent optimizations and generate the executable code. The organization in Figure 1 also shows how information from the Processor generator and even runtime profiling information could be fed back into the Machine requirements analysis phase to enable iterative refinement of the machine description, and thus of the processor itself.

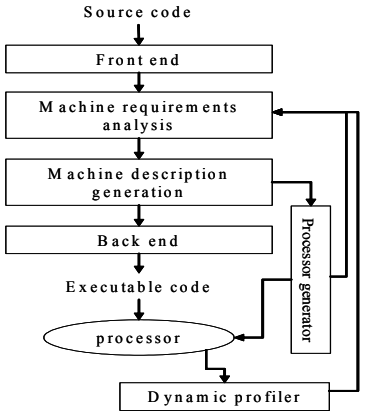

Figure 1: Organization of nanocompiler

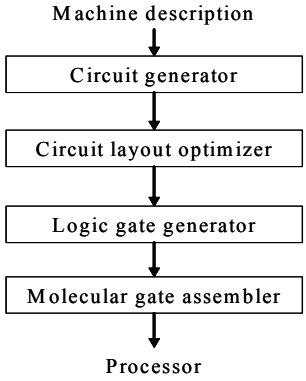

Figure 2: Organization of a nanotechnology-based processor generator

If more of the functionality of the compiler is included in the operating system or hardware, dynamic reconfiguration of the processor is possible while a program is running. Just-in-time load-time configuration involving an analysis of the requirements of a program could be incorporated into the processor's loader, as well. When a program is run on a nanocomputer with a JIT reconfiguration model, the machine could be reconfigured to better suit the program, thereby improving its performance.

Inside the **Processor generator**, either a machine is reconfigured or it is generated using a nanotechnology approach. Figure 2 shows the organization of a nanotechnology-based processor generator. The machine description is analyzed through a sequence of phases that translate the description into a layout of circuits that implement the machine, which in turn is implemented using logic gates, which are assembled using molecular manufacturing techniques, to produce the target processor.

4 Experimental Study

The prototype of our compilation approach has been implemented as an extension of the Trimaran research compiler system [22]. Trimaran was selected for its reconfigurable EPIC simulator and ease of extension. For this research, we modified Trimaran by isolating output from the live variable analysis and scheduling modules to incorporate their data into a machine description generator. Experiments in this initial study were facilitated with the use of a number of high-level control scripts and drivers that were developed as part of related ILP-targeted compiler optimization research [23].

Experiments were designed and performed to compare the impact of our nanocompiler approach on program performance. The experiments were conducted on the full set of eight C benchmarks in the SPEC95 CINT suite. Programs were compiled and run on five pre-configured EPIC processors and a 6th custom generated EPIC

processor. These six configurations consisted of 1-4 functional units (both integer and floating point), 1-4 memory units, 1-2 branch control units (to manage branch history, prediction, etc.), and registers proportional to the number of functional units (64 per). Note that the 6th custom generated processor varied in number of functional, memory and branch units. These simulated processors produce copious static and dynamic statistics, including machine cycles consumed, operations performed, memory and cache access behavior, and processor utilization. The count of machine cycles was used to calculate speedup, since the simulations in Trimaran are performed independent of any specific clock cycle frequency.

We initially expected that a carefully hand-configured general-purpose processor would provide the best overall performance for the entire suite of benchmarks. The results in Table 2 indicate that reasonable and common machine configurations do indeed produce good performance speedup. Table 2 presents speedup for four machine configurations, as well as a custom, derived, machine configuration as compared with a 6th sequential, baseline processor configuration (Machine #1, not shown).

Table 2: Comparison of speedup for various architectural configurations. Note that Machine #1, a sequential processor used for baseline comparison, is not shown.

Benchmark (lines of source)	Machine #2 2F,1M,1B Cost: 140	Machine #3 2F,2M,2B Cost: 260	Machine #4 4F,2M,2B Cost: 280	Machine #5 4F,4M,4B Cost: 520	Machine #6: Generated machine		
					Config	Cost	Speedup
099.go (29,713)	1.55	1.67	1.71	1.71	2F,1M,2B	160	1.69
124.m88ksim (20,026)	1.89	2.31	2.33	2.33	2F,2M,1B	240	2.31
129.compress (2,322)	1.95	2.23	2.25	2.74	2F,4M,1B	440	2.71
130.li (7,683)	1.30	1.39	1.39	1.40	2F,1M,1B	140	1.39
132.ijpeg (32,214)	1.91	2.11	2.22	2.32	4F,4M,1B	460	2.32
134.perl (27,768)	1.36	1.60	1.60	1.60	2F,2M,1B	240	1.60
142.gcc (211,484)	1.61	1.89	1.89	1.89	2F,2M,1B	240	1.89
147.vortex (67,286)	1.91	2.21	2.31	2.35	2F,4M,1B	440	2.35

For each configuration, numbers of functional units, memory units and branch control units are shown. For example, "4F, 2M, 1B" signifies a machine configured with 4 functional units, 2 memory units and a single branch unit. Also shown in Table 2 is a cost for each configuration, based on a cost function C, where f, b and m are the numbers of functional, branch and memory units, respectively. The cost function is:

$$C = (fxF) + (bxB) + (mxM)$$

The values F, B and M are constant weight assigned to functional, branch and memory units to attempt to assign cost based on the complexity and increased latency of increases in the numbers of each type of unit. The values assigned to these are: F=10, B=20, M=100. For example, the calculated cost of Machine #1, the sequential processor (1F, 1M, 1B), is 130.

Speedup either improved or remained steady as the cost of the simulated machine increased. Benchmarks that exhibited more memory intensive operations (129.compress, 132.ijpeg, 147.vortex) tended to continue to improve as the number of memory units increased. For benchmarks that were more computationally intensive (124.m88ksim, 132.ijpeg) or had significant data or control dependencies that constrained ILP (099.go, 134.perl, 142.gcc), additional functional or branch units tended to improve performance more than memory units.

The default configuration in the Trimaran system is Machine #3, which exhibits reasonably good speedup for all benchmarks. Results of custom generation of a machine as the result of compile-time analysis of each benchmark is shown in the last three columns of Table 2. Note that configurations tend to reflect a reduction in the number of units when additional units did not tend to improve performance. Most notably, the cost of each generated machine is less than the cost for comparable speedup provided by one of the preconfigured machines, and the Machine requirements phase appears to have frequently identified a best, or at least better, fit machine configuration for each benchmark.

Within the context of the EPIC architectures possible in Trimaran, it appears that our compiler framework leads to reduced cost, improved speedup, or a balance between the two when there are competing factors. In some cases, four memory units did not lead to speedup over a configuration with two memory units, and the machine requirements phases arrived at the correct conclusion. Within the relatively constrained domain of EPIC, VLIW,

Superscalar and similar ILP architectures, this approach may well be a feasible addition to current compilers that target reconfigurable versions of these machines.

Because a hypothetical nanocomputer architecture is quite large (billions of processing units on a single chip), the machine description generation task is more difficult. With increased flexibility and freedom of reconfiguration comes increased responsibility on the part of the compiler phases to thoroughly analyze the source program and produce a well-fitting machine description. There are improved opportunities for parallelism, and the potential to redesign the instruction set, pipeline organization, and any other feature of the machine. Ultimately, the promise of this computational nanotechnology is straightforward, on-demand molecular-level manufacturing, which means that a new processor could be specified, manufactured, used and discarded (or recycled). Although such capabilities sound far-reaching and futuristic, from a practical standpoint such a technological revolution is technically and physically feasible, if far-off. [8, 16]

5 Related Work

Research in the area of FCCMs where a system combines reconfigurable logic with a general-purpose processor has demonstrated speedup [3,13]. This research has focused on architectural details of the reconfigurable hardware and compiler features that exploit it. Our research attempts to build on this work by increasing the capability of the compiler to generate more of the target machine.

Integration of a reconfigurable functional unit into a superscalar RISC processor improves upon earlier FCCM research [4] by focusing on the hardware-compiler interface, although it relies on hand modification of source programs to achieve speedup. With our approach, all analysis and modification would be performed by the compiler.

An innovative approach to large-scale, homogeneous, undifferentiated, reconfigurable architecture improves upon FPGAs using a less expensive nanoscale "cell matrix" approach [9]. This work describes how networks of atomic-scale switches can be configured in parallel and used to fabricate scalable processors that are customized to specific tasks. Our research is targeted to a nanocomputer cell matrix architecture, focusing on use of the compiler to automatically generate reconfiguration instructions.

Hewlett Packard's research on PICO, a system that automatically designs custom computers is very similar in its goals [24-27]. With PICO, parallel portions of a program run on reconfigured FPGA hardware while the remainder runs on a connected, embedded processor. PICO employs design space exploration, selecting the best configuration from among a set of pre-designed configurations. While PICO targets primarily embedded processors and uses design space exploration, our approach envisions a desktop or embedded computer that reconfigures its own hardware to any arbitrary configuration, using machine requirements analysis and nanotechnology. This process may resemble FPGA reconfigurability, or actual physical molecular reassembly, performed at compile-time or run-time.

The field of nanotechnology is quite active and rapidly advancing, with frequent accomplishments being made in many disciplines including medicine, pharmaceuticals, chemistry, physics, and of course, computer engineering [10]. Little has been done in the area of compilers and optimization specifically aimed at using this technology. Our research attempts to pursue basic and applied nancompiler research and encourage others to do the same.

6 Summary and Future Work

Reconfigurable computing is an emerging area of hardware, and the promise of nanotechnology to hurdle looming barriers to continued applicability of Moore's Law is being recognized. We have designed a compiler framework targeted to a nanocomputer, and demonstrated that the techniques of machine description analysis and processor configuration-code generation using our compiler can improve performance while limiting costs. Although our implementation was restricted to a reconfigurable EPIC architecture simulator, as it was the most appropriate and available platform at the time of our study, there are similarities in the requirements of all processors, including nanocomputers. With the significant flexibility and capability of nanocomputers, it seems that the responsibility for guiding the configuration will fall to the compiler. This research demonstrates the feasibility of that approach. Where once the compiler customized the program to suit the machine, the compiler will soon fully customize the machine to suit the program, extending code generation to include code that will reconfigure, or even guide the manufacture of the processor itself.

We are planning extensive research and experimentation in the area of compiler optimization for nanocomputers. We are currently developing a compiler analysis framework that simulates generated architecture specifications

using Colored Petri Nets. Additional work is being done in compiler approaches to generating and simulating nano-mechanical computational machine descriptions from source code, applications to low-power and embedded computing, automatic parallelization techniques and description of nanocomputer architectures.

References

[1] D. F. Bacon, S. L. Graham, and O. J. Sharp. Compiler transformations for high-performance computing. *Computing Systems*, 26(4):345–420, 1994.

[2] P. Beckett, and A. Jennings. Towards nanocomputer architecture. Seventh Asia-Pacific Computer Systems Architecture Conference (ACSAC 2002), 2002.

[3] T. J. Callahan, J. R. Hauser, and J. Wawrzynek. The Garp architecture and C compiler. *Computer*, 33(4):62-69, Apr. 2000.

[4] J. E. Carrillo, and P. Chow. The effect of reconfigurable units in superscalar processors. *FPGA 2001*, Feb. 11-13, 2001.

[5] M. Cintra, and D. R. Llanos. Toward Efficient and Robust Software Speculative Parallelization in Multiprocessors. In Proceedings of ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, June 2003.

[6] C. P. Collier, and et al. Electronically configurable molecular-based logic gates. *Science*, 285:391-394, 1999.

[7] C. Compton, and S. Hauck. An introduction to reconfigurable computing. *IEEE Computer*, April 2000.

[8] K. E. Drexler. Nanosystems: Molecular machinery, manufacturing and computation. Wiley & Sons, Inc. 1992.

[9] L. J. K. Durbeck, and N. J. Macias. The Cell Matrix: an architecture for nanocomputing. *Nanotechnology*, 12:217-230, 2001.

[10] Foresight Nanotech Institute. Web site at http://www.foresight.org, accessed June 2005.

[11] P. Ghosh, R. Mangaser, C. Mark, and K. Rose. Interconnect-dominated VLSI design. *Proceedings of the 20th anniversary conference on advanced research in VLSI*, 114-122, Mar. 1999.

[12] W. Gibbs. A Split at the Core. *Scientific American*, pages 96-101, November 2004.

[13] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Roe, and R. R. Tyalor. PipRench: A reconfigurable architecture and compiler. *Computer*, 33(4):70-77, Apr. 2000.

[14] J. L. Hennessy, and D. A. Patterson. Computer architecture: a quantitative approach. Third edition, Morgan Kaufmann Publishers, San Francisco, 2003.

[15] IBM. Blue Gene to tackle protein folding grand challenge, available online at http://www.research.ibm.com/bluegene/press_release.html, 1999.

[16] R. C. Merkle. Design considerations for an assembler. *Nanotechnology*, 7(3):210-215, 1996.

[17] M. Montemerlo, C. Love, G. Opiteck, D. Goldhaber-Gordon, and J. Ellenbogen. Technologies and designs for electronic nanocomputers. The Mitre Corporation, http://www.mitre.org/technology/nanotech, 1996.

[18] G. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114-117, April 19, 1965.

[19] H. Moravec. When will computer hardware match the human brain? Journal of Transhumanism, vol. 1, http://www.transhumanist.com/volume1/moravec.htm, 1998.

[20] M. C. Roco. Government nanotechnology funding: an international outlook. National Science Foundation, accessed at http://www.nano.gov, June 2005.

[21] G. L. Timp, R. E. Howard, and P. M. Mankiewich. Nano-electronics for advanced computation and communication. Nanotechnology. G. L. Timp (ed). New York. Springer-Verlag, Inc. 1999.

[22] A. Nene, S. Talla, B. Goldberg, and R. M. Rabbah. Trimaran – an infrastructure for compiler research in instruction-level parallelism – user manual, 1998. http://www.trimaran.org. New York University.

[23] T. Way, B. Breech, W. Du, and L. Pollock. Evaluation of a region-based partial inlining algorithm for an ILP optimizing compiler. IASTED PDCS 2002, pages 705-710, Cambridge, Mass., November 2002.

[24] S. Abraham and B. Rau. Efficient design space exploration in PICO. CASES 2000, San Jose, 71-79, Nov. 2000.

[25] B. So, M. Hall and P. Diaz. A Compiler approach to fast hardware design space exploration in FPGA-based systems. ACM PLDI 2002, Berlin, Germany, June, 2002.

[26] K. Sekar, K. Lahiri and S. Dey. Dynamic platform management for configurable platform-based system-on-chips. Proceedings of the International Conference on Computer Aided Design (ICCAD 2003), 641-648, Nov., 2003.

[27] V. Kathail, S. Aditya, R. Schreiber, B. R. Rau, D. C. Cronquist, M. Sivaraman. PICO: Automatically designing custom computers. IEEE Computer, 39-47, Sept. 2002.