Compiling Mechanical Nanocomputer Components

GJCST Computing Classification
D.2.2, C.5.3 & B.1.4

Thomas Way, Tao Tao
and Bryan Wagner
Department of Computing Sciences Villanova University
Villanova, PA, USA thomas.way@villanova.edu

Abstract- **Computer component fabrication is approaching physical limits of traditional photolithographic fabrication techniques. Alternative computer architectures are being enabled by the rapidly maturing field of nanotechnology, and range from nanoelectronics and bioelectronics to nano-mechanical computational machines and other nanoscale components. In this study, the design of a nanocompiler, which targets a simulated hydrocarbon assembler, is presented. The compiler framework demonstrates the feasibility of a hardware compiler to produce building block components, a necessary first step in full molecular assembly of nano-mechanical computers. As a proof of concept, the resulting nano-mechanical machine components are simulated using a Colored Petri Net model of a 32-bit adder and an atomic-level gate simulator. Performance and size bound estimates and key nano-mechanical component design issues are given.**

Keywords- Nanocompilers, Mechanical logic gates, Compiler-directed mechanosynthesis, modeling.

I Introduction

Despite recent significant advancements in feature size to 45nm, announced in 2007 by Intel, there is growing consensus that the familiar density-doubling prediction of Moore's Law as it concerns 2D fabrication techniques is reaching limits [1,2]. The trend clearly is away from techniques that fit increasing numbers of transistors onto a chip, as manufacturers pursue technology that enables increasing numbers of processing cores on a single chip [3]. The movement toward increased coarse-grained parallelism, the seeming approach of inherent limits of photolithographic techniques, and the continued maturation of the field of nanotechnology could hint at a serendipitous convergence of needs and capabilities.

The way forward in chip design and fabrication may well include applied computational nanotechnology as originally foreseen by Eric Drexler [4], and furthered by many others [5-8]. Current chip design techniques, and in fact virtually all software and hardware design of any significance, make use of a variety of automated compiler tools to generate complex designs, layouts or executable code from an original human-readable specification or source program [2]. Molecular manufacturing, and other applications of nanotechnology, are likely to require a similar approach in order to manage the scope and complexity of translating a high-level processor specification into nanoscale components. Computer science techniques, such as compiler design, optimization and software engineering, are likely to play an important role in the molecular design and fabrication process. [4]

\

Modern compilers for high performance computer architectures apply a sequence of sophisticated analyses and optimizations to translate a source language program into efficient binary machine code. Machine specific optimizations, customized to the particular target architecture, are required to achieve significant speedup on modern, high-performance architectures [2,9]. In spite of public announcements made in early 2007 of advances in feature-size reduction by Intel and AMD, heat dissipation and barriers of physics remain as problems [1,5]. Nanotechnology, manufacturing performed through manipulation of atoms and molecules, or through other nanoscale manufacturing techniques, is capable of overcoming these barriers [4]. The continuing trend toward flexible computer architectures with higher degrees of parallelism suggests that the field of reconfigurable computing, perhaps enabled through the use of nanotechnology, is the next evolutionary step in processor design [5,10].

Nanocomputing is taken to mean the class of highly reconfigurable, nanotechnology-based, computer architectures, and a nanocompiler is the software-hardware system that targets such a nanocomputer. In this paper, we present the design of a nanocompiler framework that targets one form of nanoscale computer architecture, nano-mechanical computing devices. This compiler framework translates a source code program into both an optimized executable program and a customized nanocomputer on which the executable program will be ideally suited to run. Much as traditional compilers customize the program to suit the machine [9], this proposed compiler customizes the machine to fit the program. Since no such nanocomputer architecture yet exists, our study demonstrates the approach using a molecular design language, a simulated hydrocarbon assembler, and a mathematical modeling tool to demonstrate and estimate the performance of this approach.

II Background

Nanocomputer architectures, which form a subset of reconfigurable architectures that include FPGA, FCCM, cellular array, synthetic neural systems and many others [6,10], are produced using some form of a molecular manufacturing and provide a natural successor to current general-purpose microprocessor architectures [6]. To be accepted, nanocomputers functionally must be at least as capable as their predecessors, fast, inexpensive, robust and capable of operating at room temperature and of executing legacy code [5]. In order to produce such nanoscale architectures, molecular manufacturing and

mechanosynthesis techniques must be understood, including the use of hydrocarbon assemblers. As this technology is not yet realized, the use of a variety of research tools to model and simulate the proposed architecture is required.

The capabilities of various nanotechnologies as applied to science and manufacturing generally are classified into one of a range of generations or stages. Roco describes four generations [11] of nanotechnology development: passive nanostructures, active nanostructures, systems of nanosystems and molecular nanosystems. Hall defines a set of five stages [12] of increasing precision, complexity and difficulty: bulk process chemistry, molecular self-assembly, cellular-scale machinery, special purpose macro-scale molecular assembly, and general molecular manufacturing. Current advancements in nanotechnology are in Roco's second generation or Hall's second stage. The research reported in this paper assumes advancements will continue forward to at least Roco's third generation and Hall's fourth stage.

A. *Nanoscale Computer Architectures*

Among the proposed nanoscale computer architectures are silicon-based resonant-tunneling devices (RTDs) consisting of tunneling diodes paired with field-effect transistors (FETs) [5], carbon nanotube semiconductors [13], diamondoid carbon transistors synthesized using Chemical Vapor Deposition (CVD) techniques [14], a variety of quantum, DNA-based and single-electron transistor (SET) nanoscale electronic devices [7], and novel nano-mechanical computing devices constructed of moving, nanoscale components [4]. Such nano-mechanical computers are reminiscent of Charles Babbage's Analytical Engine [2], albeit nine or ten orders of magnitude smaller. Although these potential technologies hold promise, all but nano-mechanical devices appear to be constrained to two dimensions, limiting their potential for improvement over time in the same way that current photolithographic techniques are limited. The recent press release announcement regarding the maturation of IBM's chip-stacking technology may hint at a future for 3D electronic devices, although issues of resistance, heat dissipation, interconnectivity and quantum tunneling remain as significant hurdles [4,7,12].

Nano-mechanical devices such as logic gates and registers will be constructed from a series of molecular logic rods called *interlocks* and driven by kinetic forces [4]. Although this atomic-scale computational machinery will operate more slowly than traditional electronic devices, the difference will be only approximately one order of magnitude due to its much smaller size. Nano-mechanical devices can have much higher densities since interlock logic gates can be stacked in three dimensions while transistors must be placed on a 2-dimensional substrate. The density of nano-mechanical devices is estimated to be 10^{11} greater than that of silicon transistor devices, enabling the very real potential to produce massively parallel networks of nanoscale processors [4]. Other benefits to nano-mechanical computing architectures include precision, tolerance to physical wear, and improved fault tolerance [15].

The interlocks used for logic gates as described in Drexler's architecture consist of sliding rods that have knob protrusions which slide between one another. Depending on their position, one rod may block another or allow a rod to continue sliding along its vector of movement, with input and output provided by additional interlock rods, all enclosed in a stiff housing. Rods can be combined easily to form a logic gate that computes the output of a NAND operation, a logical component from which all other logical operations can be constructed. Estimates for clock speeds of nano-mechanical logic gates are 1000 MIPS, or approximately 1 GHz. The ability to fabricate in three dimensions means that massively parallel processors could be formed in very dense volumes. A complete nano-mechanical CPU system of 106 logic gate "transistors" forming a cube 400 nm on a side, with each logic gate being smaller than a single rhinovirus, would fit in the same surface area as just 80 transistors at the 45 nm scale. [4]

B. *Mechanosynthesis And Hydrocarbon Assemblers*

Building molecular mechanical computers with atomic precision will require direct control of the chemical reactions that occur between atoms and molecules. The process of manufacturing machinery through such methods is known as Molecular Manufacturing [14]. So far, chemistry has relied on methods of controlled, probabilistic reactions resulting from collisions between masses of atoms in order to synthesize useful compositions, but without guarantee of atomic precision. Mechanosynthesis is performed using atomically precise tools that rely on chemical bonding to produce positional control of mechanical forces [16], which would be used to synthesize useful molecules from their constituent atoms, as well as higher-order structures [14].

Ideally, mechanosynthesis would be performed using a Universal Constructor, a nano-mechanical computer controlled machine that could follow sequences of instructions to assemble raw atomic material into arbitrary molecular structures, including exact copies of itself [15]. Drexler's proposal for a general-purpose molecular assembler is an example of such a constructor [4,15], an approach that is reminiscent of Alan Turing's notion of a Universal Machine which eventually led to the modern, stored-program computer [2]. Research has focused on the hydrocarbon assembler, which is a simplified universal constructor that builds diamondoid structures [4], including copies of itself. The small molecular computers in each assembler are controlled by a broadcast mechanism emanating from a macroscale computer. Instructions from the macroscale controller in such a broadcast architecture are passed through acoustic waves that vibrate through a liquid environment surrounding the assemblers. The instructions are received by mechanical pressure-actuated devices in the assemblers. These pressure activated devices initiate instructions to the mechanical logic units in the

assembler computer. As a result, the assembler can be reprogrammed to make different molecular structures [8].

C. Modeling Of Nano-Mechanical Structures

Molecular machinery can require billions or trillions of atoms in the final design. Thus, there is a need for a set of "molecular compilers," which take a high level of abstraction input for molecular components and transform them into atomically positioned devices [10]. To enable high-level specification of nano-mechanical structures, an abstract scripting language, MolML, was developed that can be compiled into hydrocarbon assembler instructions [17]. MolML is an Extensible Markup Language (XML) based language that is designed to factor out the redundancy in molecular structures that have large amounts of symmetry and repetition. The language is written to provide communication between a macroscale computer and a molecular assembler, facilitating the correspondence of input instructions to the assembler. Using MolML, it is possible to define 3-dimensional molecular structures of arbitrary complexity [17].

In order to visualize and simulate the molecular structures defined with MolML, a software tool was developed to parse, display and animate a MolML design. The MolSim tool [17] implements basic Newtonian Laws by using vector geometries, accounting for the position, velocity, and acceleration of particles in the simulation environment. The simulation engine was tested against a MolML document that defined the structure of Drexler's nano-mechanical NAND gates, using the tetrahedral lattice molecular structure of rigid diamond molecules to form the logic rods. For efficiency concerns, the housing was omitted. The molecular design was inspired from an examination of the repetitive symmetries inherent in the lattice [6].

Simulation of atomic interactions and molecular structures in the current version of MolSim is infeasible for larger structures, so these larger structures such as adders can be constructed as formal, mathematical models using Colored Petri Nets (CPNs) [18]. Petri Nets provide a modeling language that is well suited for larger systems, drawing on the power of generalization provided by mathematical modeling techniques. In practice, a CPN model is created using a graphical tool, enabling visual representation and analysis of a CPN model. With the addition of timing parameters, CPN can realistically model the function of a nano-mechanical NAND gate and higher-order molecular structures constructed hierarchically by reusing the NAND model in varying configurations.

D. Nanocompiler Design

The two major goals of the work reported in this paper are refinement of the design of a nanocompiler [19] and a simulation of a common nano-mechanical component that can be generated by the nanocompiler. Our nanocompiler design generates as its output both as an executable version of the original source program and the description of a machine on which the executable will run. 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 is a generalization of compilation for reconfigurable computing in that the configuration of the target machine is unknown when compilation begins. The machine configuration is extracted from the source program, based on analyses of program characteristics. In this way, the resulting machine is an excellent fit to the program.

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 Processor generator and even runtime profiling information can feed information back into the Machine requirements analysis phase to enable iterative refinement of the machine description, and thus of the processor itself.

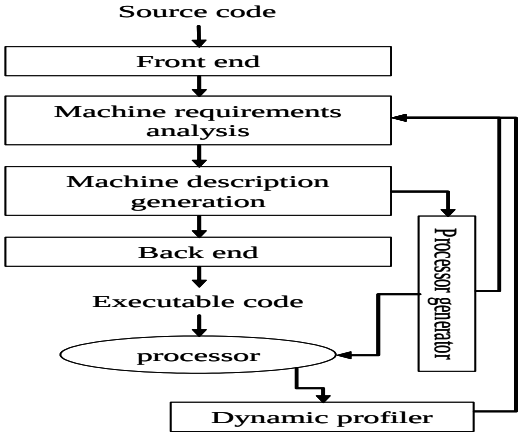

Fig.1. Organization of a nanocompiler

Inside the Processor generator (Figure 2), a machine is reconfigured or generated using a nanotechnology approach. The machine description is analyzed through a sequence of phases that translate the description into a layout of circuits (e.g., VHDL) or other structures (e.g., MolML) that implement the machine, which in turn is implemented using logic gates, which are either reconfigured as with FPGAs or assembled using molecular manufacturing techniques, to produce the target processor.

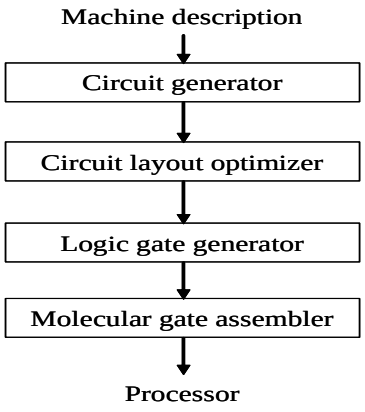

Fig.2. Organization of a nanotechnology-based processor generator.

In the current implementation, we focus on the generation of a nano-mechanical machine description for an addition instruction. Thus, we restrict the scope of the research results reported in this paper to the machine description and processor generation phases, and more specifically on how a machine description for low-level components, nano-mechanical NAND gates, can be combined into a higher-level component, a 32-bit nano-mechanical adder. Designing nano-mechanical computer components at this scale raises a number of important issues that may need to be addressed in future research, including

A. Compilation Time

Determine the time needed to analyze source code, produce a machine specification, and fabricate components using hydrocarbon assemblers, which can enable near-exponential assembly capability through an initial self-replication process.

B. Design Feasibility

Determine whether the approach is feasible, and identify further what technology must be developed to construct a complete, working nanocompiler. Extend 2-D fabrication to an understanding of 3-D fabrication made possible with mechanical components.

C. Performance Efficiency

Determine the efficiency of inherently slower nano-mechanical processing, evaluate for potential speedup through increased parallelism, and predict throughput and related latencies. Design realistic models that reflect predicted behavior.

D. Requirements Analysis

Determine what essential information is needed in the compiler, including analyses performed for parallelization, resource requirements, and efficiently scalable reconfiguration and fabrication.

E. Usability Of Massive Parallelism

Identify approaches that can successfully utilize the extreme parallelism that may be available, including integration of ILP, thread- and task-level parallelism.

III SIMULATION OF NANOSCALE COMPONENTS

Simulation of a 32-bit nano-mechanical adder was accomplished through a sequence of four developmental stages: nano-mechanical NAND gate, CPN model of a NAND gate, CPN model of a 1-bit adder, and a CPN model of a 32-bit adder. In the first stage of development, an atomic-level nano-mechanical description language (MolML) and a simulation tool (MolSim) were designed, enabling a realistic, visual modeling of a single NAND gate (Figure 3). By extending earlier work on MolSim [17], we created an atomic and molecular description of a NAND gate that could be generated given a few positional parameters, which is then translated by MolSim into the full atomic and molecular description that provides details of the placement and configuration for each atom and molecule.

Based on proposed functional parameters of nano-mechanical logic gates [4], a variety of estimated characteristics are provided in Table I. The rod component of the NAND gate can be constructed using relatively few Carbon atoms, while the housing which encloses the gate (not shown) requires significantly more. Atomically precise modeling tools are needed to quantify these material requirements. The proposed speed for a nano-mechanical processor is approximately 1 ns per operation, operating at 1000 MIPS, or 1 GHz, with logic and arithmetic operations requiring about 1 ns [4]. We assume 0.1 ns per NAND calculation, with 0.2 ns for reset latency. The volume of a single NAND gate is approximately 16 nm3 [4], so 6.25e+16 such components can fit in a single cubic millimeter. Clearly this is a much higher density than the current best practical feature size of 45 nm.

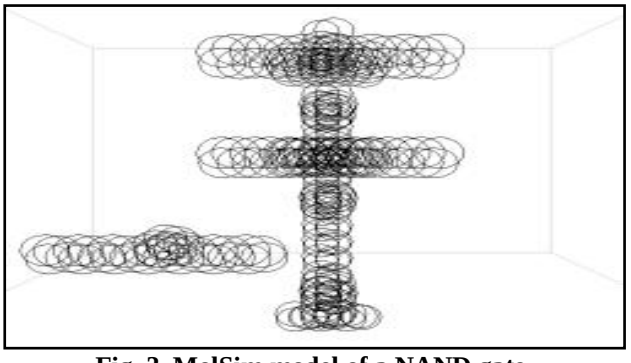

Fig. 3. MolSim model of a NAND gate.

Table I. Estimated characteristics of mechanical NAND gate model [4].

Characteristic	Estimated value
Time to perform operation (Transistors require 0.01 ns)	0.1 ns
Time to reset (2x op time)	0.2 ns

Surface area of gate (Transistors each require 10^6 nm^2)	4 nm^2
Volume of gate (gates are stackable in 3D)	16 nm^3
Improvement in volumetric packing density compared to transistors	$>10^{11}$

In the second stage, CPN was used to model the same NAND gate at a higher level of abstraction (Figure 4), but with identical behavior characteristics. Time units are measured by the addition of a timestamp notation to the NAND gate, such that the function of each NAND gate accounts for one unit of time. Time units accumulate during the simulation, enabling straightforward scaling to other time units as needed. Because of the nature of the binary calculations being performed, the NAND gate was designed to operate on Boolean, rather than decimal, values.

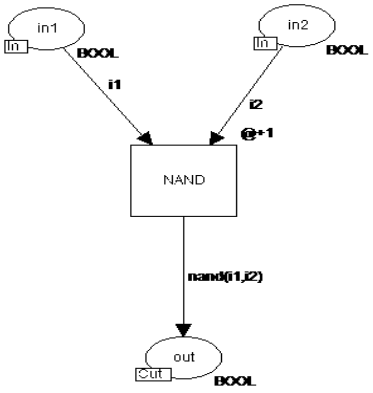

Fig. 4. CPN model of a NAND GATE.

In the third stage, a CPN model was constructed of a 1-bit adder by connecting nine NAND gate models in a standard adder configuration (Figure 5). Time measurement of the NAND gate model produced output of a carry value in 5 units of time, and a sum value in 6 time units, which equates to a total time of 0.6 ns to perform a 1-bit addition.

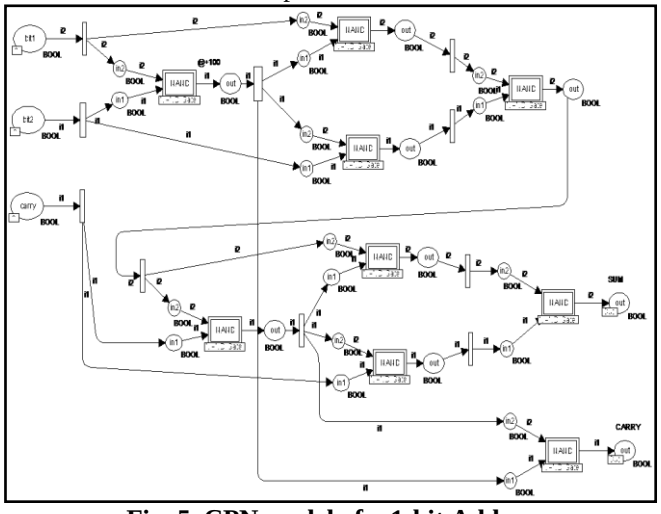

Fig. 5. CPN model of a 1-bit Adder.

In the fourth and final stage, a CPN model of a 32-bit adder was constructed using the hierarchical features of CPN Tool, combining four 1-bit adders into a 4-bit adder, then four 4-bit adders in a 16-bit adder, and finally two 16-bit adders into a 32-bit adder (Figure 6). Timing for addition of two 32 bit values was a total of 68 time units, which is generalized to the equation: 2(n-1)+6, where n is the number of bits. Although the carry value requires 5 units to calculate in each individual adder, because calculation of the sum does not require the previous carry for the first 3 time units, significant overlapping (i.e., parallel) computation occurs.

Table II summarizes selected characteristics, providing initial estimates. Based on our models, and other proposed characteristics [4], a 32-bit nano-mechanical adder requires 288 NAND gates connected using roughly 96 connector rods, possibly constructed of nanotubes in a housing. The models predict 6.8 ns per addition, with latency between the start of subsequent additions of 7.0 ns, including the 0.2 ns reset time. Drexler's proposed CPU requires approximately 1 ns per operation [4], suggesting that some amount of parallelism may be inherent in a nano-mechanical adder that is not captured in the current model. At this estimated size, 2e+14 such adders would fit in a volume of one cubic millimeter. Extrapolating these values, the presented 32-bit adder model is within specification of a proposed nano-mechanical CPU that would be contained in a cube 400 nm on a side [4].

Table II.
Estimated characteristics of a 32-bit nano-mechanical adder model.[4]

Characteristic	Est. value
Total NAND gates @ 16 nm^3	288
Total connectors @ 1 nm^3	96
Estimated volume (approx 16 nm cube) (288 x 16 nm^3) + (96 x 1 nm^3)	4,704 nm^3
Estimated time for 1-bit addition	0.6 ns
Time to perform 32-bit addition based on simulation	6.8 ns
Predicted time of 32-bit addition [6]	1 ns
Estimated throughput of model	142 MIPS
Predicted throughput [6]	1000 MIPS

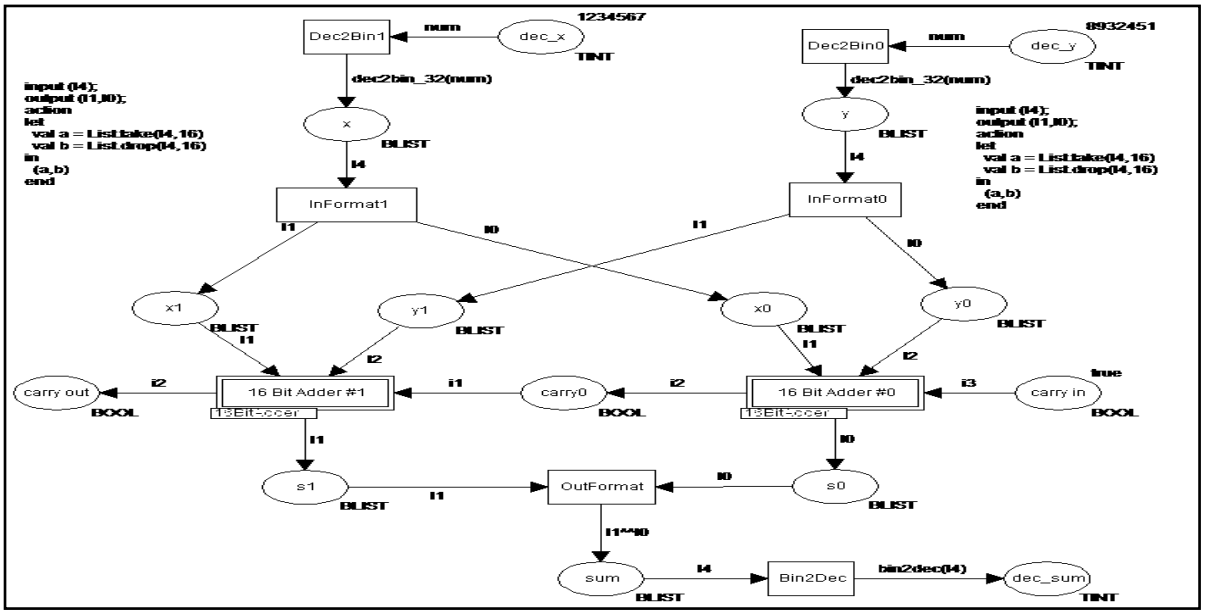

Fig. 6. CPN model of a 32-bit Adder

IV Related work

Reconfigurable architectures, including nanocomputing, are likely to provide a better fit and allow continued performance improvements for general-purpose computation. [5,6,10,20]. Design space exploration as applied to compilation for FPGA-based and other reconfigurable architectures demonstrates the performance improvements possible with customized architectures [21-23]. 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 [21]. 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 physical molecular reassembly, performed at compile-time or run-time.

An innovative approach to large-scale, homogeneous, undifferentiated, reconfigurable architecture improves upon FPGAs using a less expensive nanoscale cell matrix approach [6]. 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 can be targeted to a nanocomputer cell matrix architecture, focusing on use of the compiler to automatically generate reconfiguration instructions.

Although recent advancements in nanotechnology center on medicine, pharmaceuticals, chemistry, physics and computer engineering [12], most are early-generation nanotechnology. Our research pursues basic and applied nanocompiler research, focusing on applied molecular manufacturing techniques. Continued research in this area in the near future will require more powerful modeling and simulation tools.

Although CPN provides a flexible framework for modeling, true molecular models that more accurately depict the underlying physics and enable large-scale simulation are needed, though they will be computationally expensive. A variety of tools are available for molecular modeling and simulation (i.e., nanohub.org), with a promising recent tool being Nanorex's NanoEngineer (i.e., nanorex.com).

V Conclusion and future work

Reconfigurable computing is a rapidly advancing area, and the early promise of nanotechnology is being recognized. In this paper, we have proposed the design of a nanocompiler framework and demonstrated how molecular computing components can be generated from a higher-level representation, such as source code. Although this research is limited to simulation of a nano-mechanical adder component, the same approach can be used to generate the full range of components needed to construct an arbitrarily complex nano-mechanical multi-processor. With the significant flexibility and capability of nanocomputers, it is likely that the responsibility for guiding the configuration will fall to the compiler, and this research demonstrates the feasibility of that approach. Rather than the compiler customizing the program to suit the machine as in traditional compilation, the compiler may customize the machine to suit the program, extending code generation to include the ability to reconfigure the processor or guide its design and fabrication.

We are conducting extensive research and experimentation in the area of nanocompiler design and molecular nano-mechanical machine generation, including program characterization analysis, automatic parallelization, and molecular modeling. The long-term goal of this research is compiler control of physical, molecular assembly hardware

such as STMs and their descendants. Early work with an 8-bit adder model that uses a carry-lookahead approach [24] indicates that practical performance improvements are possible. Models for other components, including logic units, multipliers, and memory, are being developed using Colored Petri Nets for to conceptualize design and behavior and NanoEngineer to visualize the structure and physics of these nanoscale components.

VI References

1) Gibbs, W. (2004) A Split at the Core, Scientific American, 291, 96-101.
2) Hennessy, J. L. and Patterson, D. A. (2006) Computer architecture: a quantitative approach, fourth edition, San Francisco: Morgan Kaufmann Publishers.
3) Held, J., Bautista, J. and Koehl, S. (2006) From a few cores to many: a tera-scale computing research overview, Intel white paper.
4) Drexler, K. E. (1992) Nanosystems: molecular machinery, manufacturing and computation, New York: John Wiley & Sons, Inc.
5) Beckett, P. and Jennings, A. (2002) Towards nanocomputer architecture, Proceedings of the Seventh Asia-Pacific Conference on Computer Systems Architecture, Melbourne, Australia, Sept. 2002, 19, Darlinghurst, Australia: Australian Computer Society, 141-150.
6) Durbeck, L. J. K. and Macias, N. J. (2001) The Cell Matrix: an architecture for nanocomputing, Nanotechnology, 12, 217-230.
7) Goser, K., Glosekotter, P. and Dienstuhl, J. (2004) Nanoelectronics and nanosystems: from Transistors to molecular and quantum devices, Berlin: Springer-Verlag.
8) Merkle, R. C. (1996) Design considerations for an assembler, Nanotechnology, 7, 210-215.
9) Aho, A., Lam, M., Sethi, R. and Ullman, J. D. (2007) Compilers: Principles, Techniques, and Tools, New York: Addison-Wesley.
10) Compton, C. and Hauck, S. (2002) Reconfigurable computing: a survey of systems and software, ACM Computing Surveys, 34, 171-210.
11) Roco, M. C. (2007) National nanotechnology initiative: past, present and future Handbook on Nanoscience, Engineering and Technology, second edition, London: Taylor and Francis.
12) Hall, J. S. (2005) Nanofuture: what's next for nanotechnology, Amherst, New York: Prometheus Books.
13) Cohen, M. L. (2001) Nanotubes, nanoscience and nanotechnology, Materials Science and Engineering, 15, 1-11.
14) Merkle, R. C. (1993) Molecular manufacturing: adding positional control to chemical synthesis, Chemical Design Automation News, 8:9-10, 55-61.
15) Merkle, R. C. (1991) Computational nanotechnology, Nanotechnology, 2, 134-141.
16) Merkle, R. C. and Freitas, R. A. (2003) Theoretical analysis of a carbon-carbon dimer placement tool for diamond mechanosynthesis, Journal of Nanoscience and Nanotechnology, 3, 319-324.
17) Wagner, B. W. and Way, T. P. (2006) MolML: an abstract scripting language for assembly of mechanical nanocomputer architectures, International Conference on Computing in Nanotechnology, Las Vegas, June 2006, 258-264.
18) Jensen, K. (1997) Coloured petri nets: basic concepts, analysis methods and practical use, EATCS Monographs on Theoretical Computer Science, Berlin: Springer-Verlag.
19) Way, T. P. (2006) Compilation for future nanocomputer architectures nanocomputer architectures, International Conference on Computing in Nanotechnology, Las Vegas, June 2006, 251-257.
20) Carrillo, J. E. and Chow, P. (2001) The effect of reconfigurable units in superscalar processors, Proceedings of the 2001 ACM/SIGDA Ninth international Symposium on Field Programmable Gate Arrays, Monterey, California, Feb. 2001, New York: ACM Press, 141-150.
21) Kathail, V., Aditya, S., Schreiber, R., Rau, B. R., Cronquist, D. C. and Sivaraman, M. (2002) PICO: automatically designing custom computers, IEEE Computer, 35:9, 39-47.
22) Sekar, K., Lahiri, K. and Dey, S. (2003) Dynamic platform management for configurable platform-based system-on-chips, Proceedings of the International Conference on Computer Aided Design (ICCAD 2003), 641-648.
23) So, B., Hall, M. and Diaz, P. (2002) A Compiler approach to fast hardware design space exploration in FPGA-based systems, ACM SIGPLAN Notices, 37:5, 165-176.
24) Cheng, F. C., Unger, S. H. and Theobald, M. (2000) Self-timed carry-lookahead adders, IEEE Transactions on Computers, 49, 659-672.