MolML: An Abstract Scripting Language for Assembly of Mechanical Nanocomputer Architectures

Bryan W. Wagner and Thomas P. Way
Applied Computing Technology Laboratory
Department of Computing Sciences
Villanova University, Villanova, PA 19085
bryan.wagner@villanova.edu
thomas.way@villanova.edu

Abstract

Sizes of computer components are reaching nanoscale dimensions, causing physical limitations to be met in traditional computer architectures. This study surveys the field of alternative nanocomputer architectures, including the nano-mechanical computational machines first proposed by Eric Drexler. A high-level XML programming language, MolML, is introduced as a scripting language for hydrocarbon assembly of mechanical nanocomputers.

Keywords: Nanocomputer, hydrocarbon assembler, scripting, simulation

1. Introduction

As Moore's Law continues to predict the trend of continually increasing densities of transistors on ever diminishing surface dimensions, components for computer architectures are rapidly approaching sizes that can be measured in nanometers, one billionth of a meter. However, as silicon transistors become measurable on the nanoscale, certain physical properties hinder their ability to function properly as they do on the macroscale. Among these well-known physical limitations are leakage, threshold voltage control, tunneling, electro-migration, high interconnect resistance, and crosstalk. These issues can restrict electrical silicon transistor functionality to the extent that future computing designs will need to consider alternative materials and architectures [1].

Nanoscale computer architectures introduce other challenges as well. Processors are still manufactured using lithographic techniques. Lithography involves the use of a printing press to stamp, or otherwise etch or inscribe, a design into a smooth surface. Computer circuitry is constructed in this manner, and defective units are discarded. Thus, there is a probabilistic factor for error in the manufacturing process. As devices decrease in size, this error becomes excessively problematic. The long-term reliability of these components is also adversely affected by diminishing size. A number of solutions have been proposed for these problems; among them is the idea of Field Programmable Gate Arrays (FPGAs), which tolerate onboard errors by reconfiguring its methods of operation. Another problem caused by reductions in size is that the interconnection wires between computational units increasingly become a dataflow bottleneck. For logical processing units, this situation is analogous to processor-memory bottlenecks [1].

As a result, nanoscale computers may need to become highly reconfigurable, locally-connected devices with processing units that operate exclusively on onboard memory. This architecture would drastically change the classical von-Neumann stored-program method of performing computations. The localized nature of this architecture will shift designs from using single processors to using multiple processor units that exploit parallelism. Architectures that could supply "a processor for every process" may be the favored designs in the future [1].

2. Nanoscale Computer Architectures

A number of solutions for nanoscale computer architectures have been proposed, including silicon-based resonant-tunneling devices (RTDs), which consist of tunneling diodes paired with field-effect transistors (FETs). Other proposals include the use of carbon nanotube semiconductors [1]. Carbon nanotubes are carbon-based molecular structures that use sheets of graphite and the spherical C_{60} molecule known as the buckyball. Halves of buckyballs constitute the ends of the tube; the shaft is formed by wrapping flat granite molecules into a cylinder [2].

Another solution involves replacing silicon transistors with diamondoid, carbon transistors. Diamond has properties that allow it to create semi-conductors that are superior to silicon semiconductors. They allow for higher

transmission speeds due to higher electron mobility, could be fit into smaller devices due to the compact tetrahedral shape of diamond, and could operate at higher temperatures and be cooled more easily than silicon transistors. They would be less expensive to manufacture due to smaller sizes and the abundance of hydrogen and carbon, and they would operate at speeds in excess of tens of gigahertz. Unfortunately, the structure of diamond is such that it cannot be created by lithographic techniques. Current diamond synthesis is performed using Chemical Vapor Deposition (CVD), which involves growing a diamondoid surface by placing the surface in an environment of highly reactive carbon molecules. Carbon bonding occurs in random locations; as a consequence, the resulting structure contains large numbers of defects. These inaccuracies make CVD insufficient to develop diamond transistors. Positional control of atoms would be much more accurate. Such engineering would require the positionally controlled removal of hydrogen atoms and deposition of carbon molecules onto the working surface [3].

Perhaps the most novel solution, the subject of this research, involves a new class of *nano-mechanical computing devices*, first proposed by Drexler in *Nanosystems* [4]. The logic components of these devices are entirely mechanical in operation, reminiscent of Charles Babbage's Analytical Engine [5], or the ternary computing machine invented by Thomas Fowler [6]. In these devices, logic gates and registers are constructed from a series of molecular logic rods called *interlocks* and are driven by kinetic forces. Such computing devices have a subtle, yet important tradeoff with more traditional electrical circuit computers. Due to the high speed of electron transfer and the kinematics of computational machinery, nano-mechanical computer devices are slower in operation than transistor-based computers. However, because the devices are atomic in scale, the difference in speed is only approximately one order of magnitude. The advantage of nano-mechanical devices over electrical devices is that nano-mechanical computing units can have drastically thicker densities. Whereas transistors must be placed on a 2-dimensional circuit board, interlock logic gates can be stacked vertically in three dimensions. These densities are estimated to be higher than silicon transistor densities by a factor greater than 10^{11}. Consequently, this architecture provides greater opportunity for a massive network of nanoscale parallel processors [4].

Other benefits to nano-mechanical computing architectures are that since the devices are built from atomically-precise molecules, the system does not wear like a comparable system on the macroscale. This feature eliminates the need for fault-tolerance. In addition, from an analytic standpoint molecular electronics are less researched than mechanical computation, thus mechanical molecular computers are more theoretically developed. Finally, mechanical computing systems have already been physically built, as in the construction of Babbage's Difference Engine, providing a foundation of conceptual feasibility for mechanical computation [7].

The interlocks used for logic gates in Drexler's architecture are 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. A logic gate has input and output interlock rods enclosed in a smooth interlock housing. The logic gates are clocked by the displacement of a driver rod which is attached to a larger "drive rod" that provides a mechanical clocking mechanism for multiple driver rods, controlled by a motor-flywheel system. The interlock driver rod is attached to the main rod by a drive spring which aids in the displacement of the main rod in the center of the housing. A constant force reset spring rests underneath the main rod so that its position is reset when the driver rod recedes. Logic rods have probe knobs to check if a path is blocked and gate knobs to block the paths of other rods. An alignment knob is placed in the center of the main logic rod to give the rod two absolute bounds of displacement. This alignment knob is typically placed in the center to maximize stability. Forces from the drive and reset springs place the rod in either extreme, allowing the rod to exist in exactly two positional states. Based on the position of the input rods, movement of the output rod is either blocked or allowed to pass. The displacement of this rod indicates the output state of the gate. These interlocking rods can be combined easily to form a logic gate that computes the output of a NAND operation. This feature is fundamentally important since it is well known that any of the other logical operations can be constructed from a series of NAND operations [4].

The displacement sliding time of logic rods is estimated to be approximately 0.1 ns. Estimates for clock speeds rate them in excess of 1000 MIPS and running at approximately 1 GHz, which is fairly competitive compared with today's processors. The real potential behind such systems is their ability to form massively parallel processors in dense volumes: a CPU system with 10^6 logic gate "transistors" could be created within a volume of 400 nm^3 [4].

3. Molecular Manufacturing and Mechanosynthesis

Fundamentally, the properties of a molecule depend on the arrangement of its atoms. Interestingly, this fact gives carbon a remarkable range of properties. In the form of graphite, carbon forms sheets which are strongly bonded within each sheet but loosely bonded between layers, allowing the layers to slide easily past each other. In diamond form atoms are arranged in a strongly bonded tetrahedral lattice, which is both very strong and very

lightweight. It has a strength-to-weight ratio that is 50 times greater than steel, making it an ideal structure for tools. Ralph Merkle, one of the pioneering engineers of molecular manufacturing research notes that "even the ability to manufacture only the highly restricted range of structures defined by the stiff hydrocarbons would usher in a revolution in manufacturing [3]." Future structures made from interwoven diamandoid fibers may be the strongest structures to ever exist. Since nano-mechanical computers are built from rigid components, diamondoid structures are perfectly suited to their composition [3].

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 [3]." So far, chemistry has relied on methods of controlled, probabilistic reactions resulting from collisions between masses of atoms. Currently, with very few exceptions these reactions are not directed with exact atomic precision. Still, chemists are capable of synthesizing a large number of useful compositions. We would be able to build multitudes of novel structures if we were able to specify the bonding sites of chemical reactions with accurate precision. Tools that would give us such precision would take advantage of the fundamental concepts of *positional control* and *self assembly*. Positional control refers to the idea that we could use some kind of mechanical "arm" to grasp and manipulate atoms and molecules to direct the sites of their bonding reactions. Self assembly is a concept from biology, which refers to the idea that a sufficiently sophisticated arrangement of molecules can copy itself given the appropriate resources of atoms. Implementing either of these concepts would allow us to build atomically precise molecular structures [8].

The bonding reactions used in such manufacturing are mechanosynthesis reactions. *Mechanosynthesis* is the invocation of chemical bonding caused by positionally controlled mechanical forces [9]. Under the right conditions, applying a force between unbonded atoms may initiate a type of chemical bond called a *covalent bond*. Covalent bonding occurs between atoms when the atoms share electrons in their outer-most electron shell, known as *valence electrons*. Atoms tend to react when their free valence electrons come into contact with other free electrons. These unbonded valence electrons are known as *radicals*. Atoms naturally seek states of stability, and the atom is more stable when all of its valence electrons are bonded with other atoms. The resulting bonds are strong, and they are typically stronger when compared with bonds known as ionic bonds which form from electrostatic forces. Covalent bonds between different types of atoms have varying strengths [10]. Positional assembly involves positional control over the bonding sites of covalent reactions. As a result of kinetic forces, covalent bonding is initiated between free radicals and the atoms are combined into one molecule [3].

4. Hydrocarbon Assemblers

In the previous discussion, we noted that both positional control and self assembly are important concepts in the molecular manufacturing of nano-mechanical devices. In fact, self-assembly is not a necessary requirement, since any chemically stable molecule can be built by positionally controlled mechanosynthesis. However, molecular structures that will have practical application, including nano-mechanical computers, will be composed potentially of billions of atoms. Pragmatically, placing each of these atoms with single positional arms is infeasible. As a result, self replication is a practical requirement for our designs to scale to useful devices. If devices can replicate themselves, the construction process can be automated and take advantage of an exponential rate of growth [8].

Ideally, we would create a design for a "Universal Constructor" that would be capable of building large, atomically precise structures by manipulating atoms and molecules. It would be able to initiate site-specific chemical reactions that could make and break atomic bonds as desired. Such a machine would be able to build molecular structures, including copies of the machine itself [7]. Eric Drexler's proposal for a general-purpose molecular assembler is an example of such a constructor [4,7]. Most research, however, has investigated a more specific *hydrocarbon assembler*, whose purpose is to manufacture diamondoid structures. The assembler itself is made of hydrogen and carbon, so the self-replication requirement can be fulfilled. Many studies explore the class of hydrocarbon assemblers because it greatly simplifies the assembler design, yet the number of structures it could possibly create would be significantly useful due to the properties of diamond and graphite [8]. The hydrocarbon assembler is a wonderful starting point, although the universal assembler could potentially synthesize materials that exceed the strength of diamond [7].

The hydrocarbon assembler is a nanoscale device that uses molecular arms to exert positionally controlled kinetic forces on atoms or molecules to coerce covalent bonding. These molecular arms have highly reactive tips to grab structural molecules and are controlled by an onboard molecular computer, composed of mechanical, interlocking logic units. The positioning devices manipulate feedstock molecules with a "well-defined set of chemical reactions" to build molecular objects within its internal environment, which is surrounded by a diamondoid wall to block out "contaminant" atoms and molecules that could cause uncontrolled chemical reactions [7]. Within

the internal environment, the location of every structural molecule or atom is known within an amount of positional uncertainty caused by thermal noise. Under this assumption, assemblers do not need any type of sensor to locate molecules. The assembler has the capability to reproduce itself, and during the self-replication process, the assembler expands the wall of its internal region, which will break off like a dividing cell, replacing the parent with two offspring [8].

The small molecular computers in each assembler are controlled by a broadcast mechanism emanating from a macroscale computer. In this "broadcast architecture," instructions from the macroscale controller 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].

5. Scripting the Assembler: MolML

Molecular machinery can require billions of atoms in the final design. Unfortunately, many of the current CAD tools for molecular simulation require manual placement of atoms. 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 [7]. In the same manner, manually programming every action of a broadcast computer to instruct hydrocarbon assembler activity would be overwhelmingly tedious. Thus, there is a need for a more abstract scripting language that can easily describe the desired molecular structure of the work piece. Such a language could be automatically "compiled" into the redundant individual commands sent to the assemblers.

In the spirit of a molecular compiler, the author presents the MolML language as one attempt to represent molecular data in a high level of abstraction. 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. The basic motivation behind use of an XML language is that the macroscale computer easily can parse or generate XML as a high-level representation for a desired nanocomputer architecture. This architecture might be designed to take advantage of the massive parallelism that is available from arrays of nanocomputer processors. A translator would take these instructions and convert them to "assembler" code, which would then be transmitted via acoustic waves to the controller computer within a hydrocarbon assembler. The assembler would then build the device. If the input instructed the assembler to build an array of nanocomputers, it might then initiate the constructed processing units to perform the computation. The output could then be sent back to the translator (using approaches discussed in *Nanosystems* [4]) and received by the macroscale computer.

MolML adheres to the requirements of XML languages, including proper nesting of element tags, matching opening and closing tags, and definition of element attributes (Figure 1). The top-most element is the `molml` element, which defines a MolML document. There is only one `molml` element encapsulating the entire document. The `molml` element has any number of `molecule` child elements. These elements are block-level and may not be nested within one another. Each `molecule` element represents a single molecule within the assembler, and may have a single `position`, `velocity`, and `acceleration` element that represent the center of the molecule, its initial velocity at time zero, and its initial acceleration at time zero, respectively. In an actual assembler, the velocity and acceleration elements may have less significance than they do for molecular simulations, but they are still included for completeness. These three elements have attributes `x`, `y`, and `z`, representing the xyz components for their respective 3-dimensional vectors. For the `position` element, the xyz coordinates are relative to the center of the assembler, which represents the <0, 0, 0> position vector. If any of these elements are left unspecified, their values default to zero. Units of measurement are arbitrary and left to the discretion of the implementer.

To define the atoms in the molecule, the `molecule` element can define any number of `atom` elements. The `atom` element has a required `mass` element, whose value defines the mass of the atom (necessary for simulation purposes), and a required `radius` element, whose value defines the radius of the atom. The units of measurement are once again arbitrary. In addition, the `atom` element has a `position` element which defines the position of the atom within the molecule by its `x`, `y`, and `z` attributes. The `position` element has a `type` attribute, which defines how the position should be interpreted. By default, it has a value of "`relative`," indicating that the position values should be added to the position values of its parent element, which in this case are the values of the position element directly inside the `molecule` element. This behavior can be overridden by supplying this attribute with a

value of "absolute," indicating that the position should be relative to the center of the assembler, disregarding the positioning of its parent elements.

The molecule can be declared to be fixed in one of three orientations using the fixed element. This element is useful to indicate that the molecule should be attached to a structure that has a limited range of movement. For an assembler, this instruction is very high level and its interpretation largely would be a design decision. In particular, it is convenient for simulations. The fixed element has a type attribute which indicates the range of motion allowed for the molecule; it may have a value of "absolute", "linear", or "planar." The "absolute" value indicates that the molecule has absolutely fixed positioning in relation to the assembler and cannot move. In a similar manner, "linear" indicates the range of movement for the molecule relative to the assembler is restricted to a 1-dimensional linear movement, and "planar" indicates that the molecule is restricted to 2-dimensional movement within a plane. The fixed element has two child elements, fixed-vector and fixed-plane, which define vectors for the "linear" and "planar" fixed types. The fixed-vector element has x, y, and z attributes that describe the linear vector defining the range of motion. For motion within a plane, recall that a plane can be completely described by a point on the plane and a vector that is normal (perpendicular) to the planar surface. Thus, the fixed-vector element has x, y, and z attributes, which define the point on the plane, and it has attributes normalx, normaly, and normalz, which define the normal vector. Either of these child elements or both may be specified, but only the values for the fixed type specified by the type attribute are used.

An optional set of event elements describe changes in velocity and acceleration to the molecule. In an assembler, a kinetic or attractive force would correspond with the timing of these events to alter the desired vectors. The event element has three child elements: the time element designates a value indicating when the event should occur, the velocity-change element defines a change in velocity, and the acceleration-change element defines a change in acceleration. Both the velocity-change and acceleration-change elements have a type attribute which can have a value of "relative," indicating that the change vector should be added to the current movement vector, or "absolute," indicating that the velocity or acceleration should change exactly to the given vector. If left unspecified, the type attribute defaults to value "relative." The xyz components of the change vector are specified by x, y, and z attributes. Again, the units of measurement are left to the discretion of the implementer. In the event that the molecule has a fixed type, the change in velocity or acceleration is projected onto the range of motion.

The most important elements in this language are the group and repeat elements. The group element is specified to collect atoms and subgroups into one collection. It has a position child element that adjusts the center of the group in the same manner as the position child element for the atom element. Its purpose is so that the repeat element can be specified for the group, which will make copies of the group along a linear path. In this manner, most of the redundant positioning of atoms can be factored out of the MolML document, instead of absolutely specifying the position of every atom. In addition to atom child elements, a group element may have any number of group child elements which can be nested in a likewise manner. As a result, an infinite depth of repetition is possible, greatly simplifying the specification of 3-dimensional structures. The group specifies one repeat child element, which contains directions for how the group should be copied. This repeat element has three child elements: repeat-vector, repeat-distance, and repeat-count. The repeat-vector element specifies x, y, and z attributes which define a linear vector that is the path along which the group should be copied. To designate the amount of spacing between repetitions, the repeat-distance element encloses a value representing the distance between centers of repeated groups. Finally, the repeat-count element has an integer value defining the number of copies of this group that should be made along the repeat vector. This number indicates the number of copies that should be made *in addition* to the original group. A direction attribute is specified for this element, which can be given values "forward" or "backward," indicating that the copies should be made along the path of the repeat vector or against the path of the repeat vector, respectively.

The last element is used to define constant force springs which are required in Drexler's nano-mechanical designs. The spring element behaves like the group element with respect to the fact that it encapsulates a collection of groups and atoms. It cannot have a direct repeat child element. This grouping of atoms acts as the platform molecule which is attached to the spring and is subject to the constant spring force. The other atoms in the molecule element outside the spring element act as the base molecule that the spring is attached to. Several child elements define necessary information for the spring. The value of the spring-constant element defines the spring constant needed to implement Hooke's Law for simple spring motion, the spring-vector element contains x, y, and z attributes defining the vector for the linear path of spring motion, the value of the spring-

`length` element defines the maximum extended position of the spring molecule, and finally the `spring-position` element contains a floating point number between 0 and 1 or a percentage indicating the displacement in relation to the length of the spring at time zero.

With this definition, it is possible to define 3-dimensional molecular structures of arbitrary complexity. The following example (Figure 1) demonstrates a complete document representing a simple cubic structure:

```
<molml> <molecule>
  <position x="10"  y="10" z="10" />

  <group>
    <repeat>
     <repeat-distance>80</repeat-distance>
     <repeat-count dir="backward">1</repeat-count>
     <repeat-vector x="0" y="0" z="1" />
    </repeat>

    <group>
      <repeat>
       <repeat-distance>80</repeat-distance>
       <repeat-count dir="forward">1</repeat-count>
       <repeat-vector x="0" y="1" z="0" />
      </repeat>

      <group>
        <repeat>
         <repeat-distance>80</repeat-distance>
         <repeat-count dir="forward">1</repeat-count>
         <repeat-vector x="1" y="0" z="0" />
        </repeat>

        <atom>
         <radius>170</radius>
         <mass>12</mass>
        </atom>
      </group>
    </group>
  </group>
</molecule> </molml>
```

Figure 1. A MolML document

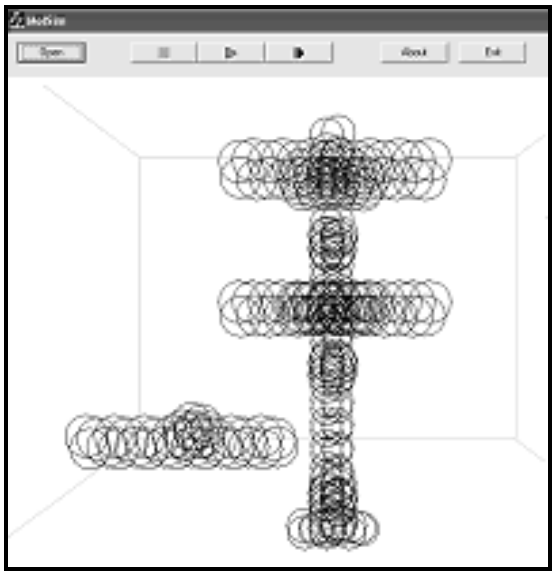

Figure 2. Simulation of NAND gate at initialization

6. Simulating the Assembled Output

To explore the feasibility of implementing the logic gates proposed by Drexler, a rudimentary molecular dynamics simulation was written (Figure 2). The simulation itself is very simple in the sense that only simple implementations of Newtonian Laws are implemented using vector geometries, accounting for the position, velocity, and acceleration of particles in the simulation environment. The simulation uses the MolML language to define molecular structures in the environment space by parsing MolML documents to initialize the state of the simulation.

The simulation engine was tested against a MolML document that defined the structure of Drexler's nano-mechanical NAND gates. The model that was used attempted to utilize 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 [9]. For each component, only four carbon atoms needed to be declared explicitly to reproduce the diamond lattice structure by using the `repeat` element. This concept allowed the resulting MolML document to greatly factor out much of the redundancy in the cubic design of the logic rods. The positions of the atoms within the repetition group was computed using the carbon covalent radius of 70 pm and 109.5 degree angles between tetrahedral carbon atoms.

Using this structure, the MolML document was parsed by the simulator. The movement of the driver, input, and output rods were controlled by `event` elements. Under the simulation engine, the mechanics of the logic gate were successfully demonstrated for all possible inputs to the NAND operation. The driver rod was able to push back the main rod which was pushing upward by force of the reset spring (Figure 2). When the input rods were in place, the driver rod was retracted and the position of the main rod was determined by the state of the input rods.

7. Conclusion & Future Work

As computer architectures progress into nanoscale dimensions, the physical limitations of silicon transistors will force architects to consider innovative computing designs, a transition that is already underway. The concept of constructing arrays of parallel processing nano-mechanical machines is particularly attractive due to their high mobility, reliability, embeddable applicability, and density which is advantageous for parallel processing networks. Since the field of nanocomputing machinery is relatively new, much research is needed to explore their design using bottom-up methods.

At the lowest level of abstraction in this nano-mechanical computer architecture, ab initio computational chemistry and molecular dynamics simulations have demonstrated the reliability of a controlled set of covalent reactions that can be invoked by mechanosynthesis. These reactions form the groundwork for an intermediate level of abstraction, in which the family of hydrocarbon assemblers are scripted to build atomically precise molecular structures. The structures defined by the MolML language are therefore a high-level schematic for scripting these assemblers. As a result, structures such as Drexler's interlock logic gates can be constructed from a simplified set of input data. The next level of abstraction would investigate the interoperability of logic gates in a complete system. This concept could develop a higher-level language for constructing and placing logical units that could be translated into MolML "assembly" instructions. The design and implementation of practical nano-mechanical machinery is feasible and should be pursued.

In addition to the continued development of MolML and associated programming and simulation tools, we are actively exploring other applications of compiler design and optimization as enabling technologies, such as program-in, chip-out (PICO), for practical use of nano-mechanical and other nanotechnology-based computer architectures. [11] Nano-mechanical computers open an entirely new realm of computing and engineering. The concept that many self-contained computers can exist on the nanoscale with the ability to replicate themselves introduces massive parallel processing opportunities. Such processing power opens the door for new approaches to classical computer architecture and algorithms, and presents the possibility for the invention of innovative embedded computing machines in everyday objects.

REFERENCES

[1] Beckett, P., Jennings, A. *Towards Nanocomputer Architecture*. Conferences in Research and Practice in Information Technology, Vol. 6, 2002.

[2] *Nanotubes and Buckyballs*. Nanotechnology Now, Apr 13, 2005. Available at: http://www.nanotech-now.com/nanotube-buckyball-sites.htm

[3] Merkle, R.C. *Molecular Manufacturing: Adding Positional Control to Chemical Synthesis*. Chemical Design Automation News, Vol. 8 (9 & 10), Sept/Oct 1993 pp. 1. Available at: http://www.zyvex.com/nanotech/CDAarticle.html

[4] Drexler, K.E. Nanosystems: *Molecular Machinery, Manufacturing, and Computation*. 1992 John Wiley & Sons, Inc.

[5] Babbage, C. *Of the Analytical Engine*. 1864. In: *Perspectives on the Computer Revolution*, by Z.W. Pylyshyn, Prentice Hall, 1970.

[6] McKay, J. *The Thomas Fowler story*. Oscar Kilo Ltd. Available at http://www.thomasfowler.org.uk/

[7] Merkle, R.C. *Computational Nanotechnology*. Nanotechnology 2 (1991), pp. 134-141. Available at http://www.zyvex.com/nanotech/compNano.html

[8] Merkle, R.C. *Design Considerations for an Assembler*. Nanotechnology 7 (1996), pp. 210-215. Available at http://www.zyvex.com/nanotech/nano4/merklePaper.html

[9] Freitas, R.A. *Pathway to Diamond Molecular Manufacturing*. Lecture, First Foresight Conference on Advanced Nanotechnology, Oct 22, 2004. Available at: http://www.molecularassembler.com/Papers/PathDiamMolMfg.htm, *Presentation discussing a four step process that will lead to the practical manufacture of nanoscale engineering tools.*

[10] *Covalent bond*. Wikipedia, the Free Encyclopedia. Available at: http://en.wikipedia.org/wiki/Covalent_bond

[11] Computational Nanotechnology group, Applied Computing Technology Lab, Villanova University, additional resources and MolSim downloads. Available at: http://actlab.csc.villanova.edu.