

Evaluation of Single Board Computers for the Antenna Controller at the Allen Telescope Array

G. R. Harp*

Allen Telescope Array, SETI Institute, 2035 Landings Dr., Mountain View, CA 94043

ABSTRACT

We review a variety of off-the-shelf single board computers being considered for application in the Allen Telescope Array (ATA) for antenna control. The evaluation process used the following procedure: we developed an equivalent small program on each computer. This program communicates over a local area network (Ethernet) to a remote host, and makes some simple tests of the network bandwidth. The controllers are evaluated according to 1) the measured performance and 2) the time it takes to develop the software. Based on these tests we rate each controller and choose one based on the Ajile aJ-100 processor for application at the ATA.

Keywords: Single board computer, embedded, Java, ATA, micro controller, Ajile

1. INTRODUCTION

The Allen Telescope Array (ATA) is a new radio telescope operating in the frequency range of 500-11200 MHz and currently under construction in northern California. The ATA's design is strongly motivated by a desire to maximize the useful collecting area for a given fixed cost, which leads to a unique design. In contrast to typical radio arrays consisting of no more than a few dozen large, expensive antennas, the ATA comprises¹ 350 smaller low cost antennas. One side benefit of this design is that the ATA will provide a generous field of view, making it an exceptional instrument for wide area radio surveys.²

The ATA control software³ is based on a paradigm of a loosely coupled network of devices. Each antenna is an autonomous device that performs its own housekeeping tasks and is commanded at a relatively high level (e.g. "follow this track on the sky"). Antennas are connected to the rest of the control system via Ethernet, and antennas may come on or go off line gracefully without interrupting activities of the rest of the telescope. To accomplish this goal each antenna is equipped with its own internet-ready computer, and this is the topic of the present paper.

The antenna computer (or controller) must support TCP/IP over Ethernet and have sufficient processing power to a) drive azimuth and elevation stepper motors with high accuracy for pointing, b) command several dozen devices at the antenna such as the low-noise amplifiers, cryogenic refrigerator, RF electronics, electronic cooling systems, etc., c) monitor all devices including tracking, refrigerator temperature, line voltage, etc and store this data, eventually returning it to the main control system for archiving. Another constraint is cost. Since there are 350 such computers, our budget allows that they should cost \$500 or less in quantity (ideally much less). With these constraints in mind, we undertook a study of a variety of single board computers and attempted to characterize them for suitability as the ATA antenna controller.

When we began to search for controllers in fall of 2001, we discovered that some candidates could be programmed in Java. This was identified as a greatly beneficial because a) the rest of the ATA control system is written in Java and b) Java's portability ameliorates some of the risk associated with choosing any one controller. For example, one difficult issue with controller software is to handle multitasking. There is not a single standard approach to multitasking (or multi-threading) in C, hence every C-based controller is programmed differently. Comparatively, controllers that implement the Java threading model are programmed in identical fashion. In the unfortunate event that a Java-based controller must be replaced with a model from a different vendor, Java multi-threading code will port more easily than equivalent code in C.

* Email: gharp@seti.org.

We have taken a very hands-on approach to controller evaluation. Rather than simply studying specifications, we believe that it is important to get experience actually using various single board computers before adopting one. Our evaluations include both objective and subjective measures and ask the following questions:

- What is the quality of the development environment? How easy or hard are they to program or reprogram?
- How mature is the underlying software (e.g. libraries or OS)? Is it bug-ridden?
- What communications efficiency can be expected?
- What processing efficiency can be expected?

To answer these questions, we developed an equivalent “sockets” program on each computer that communicates with a host computer via Ethernet. We recorded the time it took to develop each program and made notes about the programming experience. Once developed, the program provides a benchmark of the processor and communication efficiency. These and other factors were all considered in our review.

The information provided here is time-sensitive for a couple of reasons. Firstly, the manufacturer support for all these systems is generally improving with time. Secondly, new devices are appearing on the market every day. Our evaluations were mainly carried out during the winter of 2001. Furthermore, this study is not comprehensive since there was a deadline associated with choosing the controller. Yet this case study may be useful to the astronomical community by providing a snapshot of the present state of the art in low cost embedded systems.

2. CHOICE OF CONTROLLERS FOR STUDY

Because of the short development cycle for the ATA, we sought an off-the-shelf solution. A wide range of controllers was considered with prices varying from less than \$100 to \$1000. To be considered, a controller must support TCP/IP over Ethernet, have a rewritable persistent memory (flash, no disk drive), and have a minimum of on-board I/O (e.g. serial ports, digital I/O, with bonus points for hardware timers, CANBUS interface, or other options we foresee using). We put a priority on ease of use, placing high value on systems that provide mature development environments. We had a substantial bias toward controllers that support Java programming.

Table 1 lists some of the controllers that were considered along with their salient properties. For comparison, we also include the properties of a few desktop computers employed in the study.

1. Java Support

For an embedded controller the level of Java support can vary greatly, and Sun has developed various “versions” of Java appropriate for devices with differing capabilities. For example, many cell phones are programmed via the Java CLDC API (Connected Limited Device Configuration). More recently Sun developed the CDC API (Connected Device Configuration), which is aimed at more capable embedded devices. The next step up is J2SE (Java 2 Platform, Standard Edition), which is targeted to PC-level systems. Because the primary ATA control system is written with J2SE, it is desirable to have the same support in the antenna controller.

On an orthogonal development path, Sun has recently released the Real Time Java (RTJ) API, providing extensions not available in the releases mentioned above. These extensions address limitations of “standard” Java for real time work. For example, automatic garbage collection takes place as a preemptive task in standard implementations. This means that occasionally, your application halts for several milliseconds (or more) while the garbage collector is run. In embedded applications, such behavior may be unacceptable, and RTJ gives the programmer greater control over garbage collection.

All the Java controllers given serious consideration here support at least CLDC compliant Java. Our rationale is that anything less is really a dialect of Java, and would not provide the benefits of unity and portability that we seek. For example, the Javelin Stamp® (www.javelinstamp.com) is an inexpensive controller that is programmed with Java syntax and a mostly proprietary class library. In our opinion, the

overlap between the Javelin API and the standard Java API is too small to offer much in the way of portability.

A somewhat dated review of embedded Java systems (ca. Oct. 2000) can be found at http://www.mitre.org/support/papers/tech_papers_01.⁴ Even in the last 6 months several more Java-ready embedded systems have become available, with more to come. This rapidly developing market should be watched closely by those designing embedded systems.

Name	CPU	CPU/ Bus (MHz/ Bits)	RAM/ Flash (MB)	Runtime Environ.	Ethernet Adapter Speed (Mbit/s)	Serial Ports	Digital I/O (Bits)	Price at pur- chase	Misc.
Windows Laptop	Pentium III	600 / 32	256 / NA	Java 1.3	100	2	8+	\$3000	
Windows Desktop	Pentium III	450 / 32	128 / NA	Java 1.3	100	2	8+	\$2000	
RCM2100	Rabbit 2000	22 / 8	0.5 / 0.5	Dynamic C	10	2	34	\$280	cheaper in quantity
TINI / Step	Dallas Semi. TINI	18 / 8	1 / NA	Interpreted JVM	10	2	16	\$130	Has CAN, I-Button 1-Wire
Arcom	Intel 386	25 / 16	2 / 1	RTOS / C	10	3	NA	\$300	No JVM
SaJe	Ajile aJ-100	100 / 32	1 / 4	Native JVM	10	2	20	\$500	1-Wire, SPI, more
AJ100-EVB	Ajile aJ-100	100 / 32	1 / 4	Native JVM	10	2	13	\$1000	LCD, Touch Screen, SPI port
Axis	Axis Etrax 100LX	100 / 32	4 / 8	Java 1.3	100	2	16	\$450	Quantity pricing \$250.
PCM-48E26	Intel 486	133 / 32	32 / 48	Java 1.3	100	1	0	\$680	can be configured with more I/O at higher price

Table 1: Sampling of computers considered in this study.

For more information about these controllers see: (RCM2100) www.rabbitsemiconductor.com, (TINI / Step, SaJe) www.systonix.com, (Arcom) www.arcomcontrols.com, (AJ100-EVB) www.ajile.com, (Axis) www.Axis.com, (PCM-48E26) www.emacinc.com.

3. RESULTS

The controllers considered here are all “internet ready” in the sense that they provide an Ethernet connection and support TCP/IP through a “sockets” interface. This interface can be difficult to master, especially on controllers that are programmed in C since there is no standardization of how sockets are invoked or how data is transferred (e.g. byte order). For controllers programmed in Java, our experience is that sockets invocation and data transfers are more easily ported.

2. Description of the Ethernet Tests

To evaluate each controller, an equivalent small program (called “server”) was developed on each controller. This program opens a server socket and listens for incoming connections from the “client” program, running on the Windows Laptop mentioned in Table 1. The server and client communicate over a controlled laboratory network.

After the socket connection is established, a series of messages transactions are made. A single transaction consists of the client sending some data (i.e. a request) followed by a response sent back from the server. Various request / response pairs are implemented. For example, request packets include

- a single string of variable length,
- a string followed by a pair of floating point values (17 bytes), and
- a string followed by a Date object comprising several strings and integers (39 bytes).

The exercise of implementing 3 different value types was quite helpful for assessing the ease of use of the development environments.

Several communication tests are constructed on this platform:

Test_A: The socket is opened and N transactions are initiated in rapid succession. As soon as one is finished, the next one is begun. After all transactions are complete, the socket is closed. The time for N transactions is recorded and divided by N for our benchmark.

Test_B: A socket is opened, a single message transaction occurs, then the socket is closed. This cycle is repeated N times in rapid succession and timed as above. The benchmark is defined as the total time for N transactions divided by N.

Test_C: This is a variation on Test_A. Here a single string is sent to the server which echoes it back, repeated N = 100 times. The string length is varied over a wide range (4 – 8192 bytes). Using this test we can deduce the maximal data transfer rate. Here the benchmark is the one-way data transfer rate in Mbit/sec.

The client and server programs were initially developed on the Windows platform and then ported to the controllers. Because some of the controllers are programmed in Java while others are programmed in C, Windows servers were developed in both languages. (Between two desktop computers, no significant performance difference between the Java and C server was observed.) The client program was written in Java. As a very crude benchmark of usability, we estimate the time spent from the moment the controller box was opened until all servers were developed on each controller. This typically includes substantial time to become familiar with the development tools.

3. Ethernet Performance

As will be seen, the benchmarks from these tests show a great deal of scatter. The most easily interpreted results are those from Test_C. Figure 1 displays the average data transfer rate as a function of string length, L. Most curves are monotonic functions that saturate for large values of L. This is just what we expect, since the overhead associated with opening the socket and initiating each transaction should be fairly constant, and the contribution of this overhead to the transfer rate should decrease with increasing L. Note that other controllers employing the same chipset as those in Table 1 were also tested. We did not find significant differences between controllers employing the same chipset, so Figure 1 displays only six representative curves.

The implementation of the client program uses the `java.io.DataInputStream` class, which performs at least two copy operations on the string as it is read from the socket. Even between the fastest processors (Win-Desktop curve), we observe that the net transfer rate does not approach the Ethernet baud rate (100 Mb/s) hence must be software limited. Nevertheless, this is a meaningful benchmark as it exemplifies a careful but straightforward implementation of sockets communication.

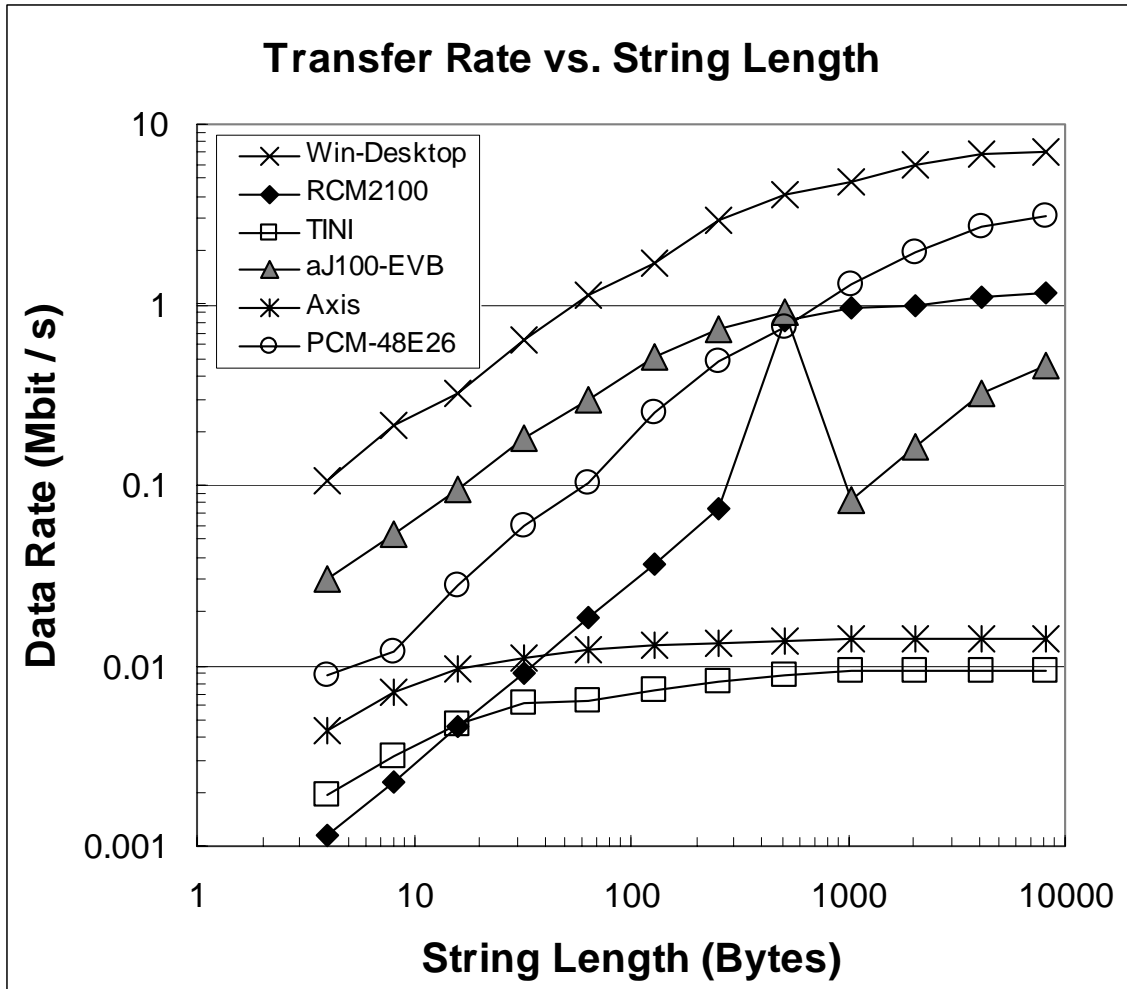


Figure 1: Data transfer rate versus the size of the data block being transferred.

Two curves show unsmooth behavior: aJ100-EVB and Axis. This probably has to do with details of the Ethernet drivers for each controller (specifically Nagle's algorithm^{*}). Since these parameters can generally be tweaked, we focus on the transfer rate of a 512-byte string as a representative sample.

Three of the controllers show transfer rates about 10 times slower than that of a Windows desktop (aJ100-EVB, RCM2100, and PCM-48E26). This is reasonable considering the lower processor speeds and 10 Base-T Ethernet connections on these controllers.

Two of these curves saturate at surprisingly low values (TINI and Axis) – 100 times slower than the other controllers. We have traced this slowdown to the difference between the servers running as native code versus interpreted code. The desktop computers and PCM-48E26 employ a Just In Time® (JIT) native

^{*} The RCM2100 data may be explained by supposing the driver delays transmission of small packets less than 256 Bytes in length in hopes that more small packets will arrive during the delay (this is one form of Nagle's algorithm). The results from aJ100-EVB might be explained by another aspect of Nagle's algorithm, where after sending one packet, subsequent packets are delayed unless the recipient returns an acknowledgement packet. We suppose that strings larger than 512 Bytes are broken up into multiple packets, introducing added delays. In any event, packet fragmentation would introduce some overhead.

[†] Note that Java 2 provides socket level control of whether or not Nagle's algorithm is applied through the `java.io.Socket.setTcpNoDelay(boolean)` method. Throughout this paper, on all Java platforms that support this method, we turned Nagle's algorithm *off* for the tests (`boolean = true`).

compiler built in to Sun's JVM. The RCM2100 is programmed in C, and programs are compiled to native in the usual way. The aJ100-EVB uses the aJ-100 processor, for which Java bytecodes *are* the native instruction set. Comparatively, on both TINI and Axis the Java bytecode is interpreted at runtime (JIT not available).[‡]

It is easily shown that the transfer rate is not hardware limited by considering that ftp transfer rates may exceed 16 Mbit/s on Axis. Similarly, on the TINI card the maximum ftp transfer rate is 0.1 Mbit/s, which is still an order of magnitude faster than our benchmark.

The results of Test_A and Test_B are presented in Table 2. Beginning with Test_A, the data show a lot of scatter, and reinforcing our observation that the transfer rate is not easily predicted. The short messages used here give us some idea of the minimum response time for communicating with these controllers. With proper tuning, any of the controllers can be coaxed into acknowledging a request within 10-56 ms.

Controller	Test_A Date (ms)	Test_A Floats (ms)	Test_B Date (ms)	Test_B Floats (ms)
Windows Desktop	7.3	2.7	14	5
RCM2100	56	56	3620	3620
TINI	531	48	8140	940
AJ100-EVB	370	10	3800	1520
Axis	102	28	183	62
PCM-48E26	72	24	243	60

Table 2: Results of Tests A and B, using either date objects or float objects. Lower scores are better.

Test_B is dominated by the time it takes to initiate a new socket connection to the controller. We see that socket initiation is expensive on most controllers, though some fare better than others. This has implications for program implementation: when communicating with most controllers, one should keep the socket open for as long as there is data to transmit – possibly forever.

4. Usability

This section presents some highly subjective observations of the author. With that caveat, here are the results.

Notice that the Arcom controller was not mentioned in the previous section. We were unable to develop using this card due to confusion and bad timing. The Arcom controller was purchased several months before the beginning of this study, and we believed our purchase included a complete kit (specifically, it included a software package called the “hardware development kit”). However, when the box was opened, we learned that we were missing a critical piece of software: the “software development kit.” Contacting Arcom, we learned that the SDK was no longer available for that controller, though a new one was due for release later in the year. We did not pursue this vendor beyond this point.

Apart from that, Table 3 presents impressions of usability for five representative controllers. Tool Quality is an evaluation of the development tools (compiler, linker, loader, debugger) that may be purchased for the controller, merging observations about learning curve and professional quality. Stability indicates impressions about whether the runtime environment is “finished” or still immature (i.e. buggy). Support rates the responsiveness of technical support. Finally, Time to Port is the approximate total work time elapsed between opening the box and completing the server applications.

[‡] We contacted Axis regarding this interpretation and they confirmed it, pointing out that developers may deploy C programs on this card and achieve significantly better performance.

Controller	Tool Quality	Stability	Support	Time to Port (days)	Overall Ease of Use
RCM2100	B+	A	C	5	C
TINI	C	C	A	3	C
aJ100-EVB	B	B	A	2	B
Axis	A	A	A	<1	A
PCM-48E26	A	A	B	2	B

Table 3: Ratings of usability for representative controllers. The grading system varies from high (A) to low (F), except Time to Port for which lower scores are better.

5. Details of Usability

RCM2100

This controller has no operating system per se. C programs are developed in a proprietary environment (Dynamic C) that supports language extensions (co-statements, co-functions) for multitasking. All development is performed with a single IDE acting as editor, compiler, linker, and symbolic debugger. The IDE has excellent quality and the runtime system seems quite stable.

The learning curve for the proprietary extensions is steep. With this unfamiliar paradigm we anticipated delays in application development. We also stumbled over byte-ordering issues during implementation of integer and floating-point communication. (Because Java explicitly specifies numeric representation, this was never an issue on the other controllers.) Finally, because of the proprietary nature of the environment, developing on this platform seems risky. If some future requirement forces us to switch to another controller, porting our code would be difficult.

There is no user group for this device, and while struggling with server development I emailed technical support a couple of times. I finally received a (useless) response more than one week after submitting my request, by which time I had solved my own problem.

TINI

On the TINI controller, one develops Java programs with any editor and uses the Sun compiler in conjunction with a proprietary class library.[§] A command-line program performs static linking.^{**} A limited multitasking Unix-like OS runs on the controller, and Java programs are executed (interpreted) on a Java Virtual Machine (JVM).

It should be stressed that TINI is by far the least expensive of the controllers considered here. So it is perhaps pardonable that the tools and runtime have an amateurish feel and do not inspire confidence. The technical support is highly responsive, and there is a very active user group. Unfortunately, much support is needed as the runtime system and development environment are still in flux. The learning curve for development is relatively high since there are many, many steps to be performed before one is ready to run the program (reset, load the boot loader, load the OS, boot the OS, compile / link / load the program, ...).

Arcom

See paragraph under Usability.

[§] A proprietary subset of the Java standard API is supported.

^{**} Ordinarily, Java programs are not linked and class resolution occurs at run time. However, run time class resolution incurs a performance penalty at class-load time, and generally requires more memory space. Static linking allows classes to be loaded more efficiently, and also allows optimization – classes and functions that are never called may be elided from the code. Both TINI and aJ100-EVB employ optimizing static linkers.

aJ100-EVB (and SaJe)

This controller supports no OS. Instead, Java programs enjoy *native* execution on the CPU. This is unique – the CPU is designed such that the native instruction set is the set of Java bytecodes. Native support for Java real time extensions (www.rti.org) and hardware implementation of the Java threading model makes it an attractive platform. Multitasking is provided via Java threads and the capability for running multiple JVM's. The controller fully supports the Java CLDC API, and a limited subset of the Java CDC API. Future releases that fully support CDC are promised.

Programs are written with any editor and compiled (Sun compiler) against Sun's CLDC classes and proprietary classes supporting hardware and CDC extensions. Two professional-quality GUI-based programs perform static linking and loading / debugging. These programs and the runtime system give an impression of stability.

After becoming familiar with the tools, program development was straightforward. Technical support is available from the controller manufacturer (Ajile), the Ajile user group, another manufacturer (Systronix, maker of SaJe) who employs the same chip, and the Systronix user group. All these groups are responsive.

Program development was slowed down by limitations of the CLDC API. For example, socket invocation in CLCD is slightly different from that in the Java standard edition. By comparison with porting between dialects of C (e.g. RCM2100), porting the servers was a great deal easier here.

Axis

Axis employs an embedded Linux OS, which comes pre-loaded in ROM. You can ask that your controller is preloaded with a proprietary implementation of the Java runtime, which has a small memory footprint. Despite its proprietary nature, we did not notice any difference between programming the Axis runtime and programming for Sun's runtime.

Installation is quite easy. Java programs developed and compiled under Windows execute *without change* on this controller. You simply ftp the class files to the controller and then run from a terminal window (terminal comes in on Ethernet port).

Regarding ease of use, the Axis controller is the clear winner. The box was opened at 8:00 am, and all the servers were running by 2:30 pm. The complete Java standard edition is supported, and development uses any tools with which the developer is familiar. The Axis sales staff was responsive to questions. Axis has a user's group, but we did not need it.

PCM-48E26

Like Axis, this controller employs an embedded Linux OS, which comes pre-loaded in ROM. A major difference between this controller and Axis is that the PCM-48E26 uses Sun's JVM. As a result, the memory requirements for PCM-48E26 are much higher. It took us a couple of iterations with the manufacturer to get the correct version of Linux and Java runtime installed, including sending back the flash memory chip for replacement with a larger one.

As with Axis, installation is easy. Java programs developed and compiled under Windows, ftp'ed to and run on the controller. Interaction with the controller is through it's one and only serial port.

The EMAC support is reasonably good. The salesman returned my calls and emails. Technical support usually took a couple of days to respond, but eventually came through with valuable information. Most of my time was spent tracking down OS issues rather than actually porting code.

The hardware support on this controller is minimal, and we imagine that JNI interfaces would have to be written to provide necessary hardware control. Based on previous experiences with JNI, this is probably not simple.

Numerical Performance

The Axis card is very attractive for its ease of use yet its Ethernet performance under Java is disappointing. Supposing that the transfer rate issue might be overcome with added development work, we constructed another performance test.

One task our controller will commonly perform is to interpolate a function evaluated at a series of discrete points. For example, the azimuth and elevation of an astronomical source might be transmitted to the controller on a 1 second time interval while the controller interpolates to an irregular time interval. We implemented a floating-point cubic spline interpolator in Java and ported this to both the Axis and aJ100-EVB cards. We measured the time required for N interpolations and record the total time divided by N in Table 4.

Name	Time for 1 Interpolation (μ s)
Windows Desktop	20
aJ100-EVB	83
Axis	4661

Table 4: Benchmark of numerical processing speed.

Here we again observe the vast difference between native and interpreted Java execution. This difference is attributed to two factors: a) the Axis has no hardware acceleration for floating point operations and b) the Axis uses interpreted Java.

4. DISCUSSION

Beginning with the RCM2100, we note that it is explicitly single-tasking, though it supports a kind of multi-threading.^{††} This makes it difficult to employ in our application since we wish to download software updates on the fly. Furthermore, software must be loaded through a proprietary programming port. Because of this and concerns cited above (portability, usability) we rejected this controller for our application.

The TINI card is likewise rejected. Major concerns are the platform stability and the controller performance. Specifically, the Ethernet performance is so low as to render the controller useless for our application.

When running Java, the usability of the Axis card makes it very attractive. However, the Java performance is problematic. Both the Ethernet and numerical performance fall below the minimum level required for our application. In principle, many performance limits can be overcome using native method calls (or simply writing the application in C or C++) but this complicates development and reduces Axis' attractiveness. Finally, we rejected this controller as well.

A strong argument can be made for adopting (a variant of) the PCM-48E26 for our application. Full-blown Java 1.2 is supported on this controller and many development steps (such as program installation by ftp) are straightforward. The main drawbacks are cost and hardware control. Notice that standard Java does not provide a standard API for digital I/O control (unless you count the parallel port). Comparatively, the manufacturers of the other controllers here all provide proprietary API's for low-level I/O control. This can save significant labor in the long run.

Ultimately, we chose the aJ-100 processor chip as the fundamental basis for the antenna controller. We are currently prototyping with SaJe boards, but another style controller, JStik, is available soon from Systronix. Development for any of these cards is practically identical, so the final choice need not be made immediately. The aJ-100 controllers support two concurrent JVM's. This feature can be used for on the fly downloads of program updates. One JVM negotiates the download and serves as watchdog timer for the system. The second JVM is started from the first and performs most of the work.

^{††} Note that RCM2100 supports a hardware watchdog timer providing one aspect of a multi-tasking environment.

aJ-100 computers support three frequency-controllable hardware timers. This provides a convenient way of controlling the azimuth and elevation axis motors. These are stepper motors, and require TTL pulses with slowly varying frequency (0-10,000 Hz) as a function of time. Rather than setting up a tight loop to manually control an I/O pin, these timers provide a simple way of driving the motors at constant velocity, allowing leisurely updates of timer frequency in the application code. This feature is not available on any of the other cards discussed here.

aJ-100 computers support the Java threading model at the native level. As a result, thread switching is *extremely* fast, only $\sim 1 \mu\text{s}$. This is fast even compared to Java programs on Windows desktops, where thread switching may take 1 ms or more.

5. CONCLUSIONS

Our main conclusion is that before committing to a particular controller, it is very important to prototype your application on real hardware. There were many surprises along the way during this study. For example, we have gotten used to Java performance on desktop machines where JIT runtimes are available and supposed that performance on controllers might be predicted simply by scaling the processor size. This calculation turned out to be wrong by two orders of magnitude! In another case, we assumed that C development would be equally difficult on all platforms but learned that some controllers (RCM2100) implement certain OS functions via proprietary C extensions, greatly complicating program development.

Other surprises included hardware limitations. One other controller we purchased but not mentioned until now, called the JStamp (Systronix), sports an RJ-45 connector. We carelessly concluded that this was an Ethernet connector (not true). In another example, on the RCM2100 programs must be loaded through a proprietary hardware port, making software updates via Ethernet impossible. This was not at all clear until we had some experience with the controller.

Regarding the use of Java in an embedded system, the state of the art is changing rapidly. In a year or two, embedded Java will be more widespread and many more options will be available. At that time we may choose to shift to a different controller, depending on whether Ajile continues to improve support as promised. Assuming it does, the aJ-100 solution is expected to remain attractive over time. Future support of the Java CDC API will simplify coding and porting, and Java real time extensions may make this controller attractive in other areas of the ATA control system. In any event, the inherent portability of Java gives us a reassuring feeling of control.

6. REFERENCES

1. J. W. Dreher, "The One Hectare Telescope", URSI-USNC Abstracts, p 33, Boulder, 1999. For a more up to date description of the ATA, please contact the author.
2. D. Bock, "350-antenna sample configurations for the Allen Telescope Array," ATA Memo **21**, <http://astron.berkeley.edu/~dbock/papers/index.html>.
3. A description of the ATA software architecture can be found in paper 4848-01 elsewhere in this volume.
4. A. Piszcz and K. Vidrine, "Real Time Java Commercial Product Assessment," Mitre Corporation (http://www.mitre.org/support/papers/tech_papers_01/piszcz_rt_java/index.shtml).