A Comparison of Aegis Signal Processing Implementations on Multiprocessor Servers and Embedded Systems

June 2002

Thomas F. Steck, Peter G. Vouras, Vilhelm Gregers-Hansen, Gerard G. L. Meyer
Naval Research Laboratory, Washington D.C., 20375 and
Johns Hopkins University, Baltimore MD., 21218

Abstract

This paper investigates the performance trade-offs inherent between multiprocessor servers and embedded systems used to perform the real-time signal processing functions required by many radars, including the AN/SPY-1 radar in Aegis combat systems. We consider some of the classical constraints such as size, power consumption, reset time, real-time operating system availability, and cost; but we also consider additional factors such as compiler sophistication, ease of programming, optimal performance versus typical performance, code portability, and software re-use. In general, we have found that multiprocessor servers offer high performance with less programming effort, but lag behind embedded processors in most of the classical constraint areas to the extent that they are difficult to justify for use on mobile platforms.

1.0 Introduction

Two multiprocessor servers and one embedded system are evaluated. The multiprocessor servers are the Hewlett Packard Superdome and the SGI Origin 3800. The embedded system is a Mercury Computer system powered by PowerPC 7400 processors with AltiVec vector processing units. The Superdome used has 48 HP PA-8600 processors running at 550 MHz, has 1 GB RAM per processor, is 77.2" x 60" x 48", and costs $1.8M. The Origin 3800 has 512 MIPS R12000 processors running at 400 MHz, and has a mixture of 512 MB and 1 GB RAM per processor. Typical price for this SGI O3800 configuration is $8.9M. The Mercury Computer has 64 Motorola 7400 processors with AltiVec units running at 400 MHz, 64 MB RAM per processor, is 19" x 22.75" x 28", and costs $1M. None of the systems evaluated had real-time operating systems installed; however, the Mercury System guarantees that no other jobs are running on the system at the same time by allowing only one job at a time to attach to a processor.

The rest of this paper is organized as follows. The first section lists the various criteria used for comparing multiprocessor servers and embedded systems. The next section details the signal processing problem used to test each platform. Performance related results are then provided with some analysis. There is some additional work required to port the software to embedded systems, especially if performance is an issue. This problem is discussed further in the third section. Finally,

we summarize with some thoughts about the role of multiprocessor servers and embedded systems in future real-time signal processing applications.

2.0 Comparison Criteria

There are a number of factors to consider when comparing computing platforms. Classically, software engineers and system integrators have looked at hardware features. A great deal of this information can be gleaned by looking at the manufacturer's data sheet for the product. But, rarely has a company been able to deliver a product that delivers the full processing performance advertised. In fact, the norm has been to expect somewhere around 15% to 25% of the peak processing performance. Expert software designers may be able to push parts of their code to near peak performance, with an overall performance approaching 70% or 80% of peak; however, the time, effort, and expertise required are guaranteed to be substantial. Therefore, the software development environment is an important area for comparison. Unfortunately, as important as this area is, it is often difficult to quantify. Nevertheless, some qualitative observations can be made.

In the world of high performance computing, the latest and greatest machine very seldom lasts beyond even a software design cycle. In addition, the processing specified for a given task is continuously refined. Therefore, the software must be relatively portable and robust to changes in the design specifications. On high performance computers, software portability is affected by both hardware and software. Finally, no matter how capable a system is, if it is unable to meet the specified latency and throughput requirements, it is unacceptable.

The CPU chosen for a multiprocessor system is the single most important piece of hardware. In addition to the presence of one or more processing pipelines, it may also contain vector processing units, such as the Motorola PowerPC 7400 with AltiVec that is used in the Mercury Computer systems. The CPU may also provide support for multiprocessing by incorporating hardware such as a cache manager that checks to see if a remote processor has updated either its cache or the main memory, thereby requiring a local cache update. Processor speed, rated in MHz or GHz provides some measure of performance, but often fails to be a good measure when comparing processors from different product lines or different companies. Related to processor speed is power consumption, which in turn, affects the system operating temperature. In a tightly packed system, again using the Mercury as an example, critical system components fail to operate at high temperatures caused by the heat from multiple processors and an operating environment that makes heat dissipation difficult. The maximum number of processors in a system may be determined as much by thermal and power considerations, as by the ability of the interconnection network to handle the data traffic.

When it comes to performance, effective use of the memory hierarchy is critical to obtaining the best results. Data that is necessary for ongoing calculations should be already available in the registers or the first level cache, where access times are typically a few clock cycles. Once data has been returned to main memory from cache, it should be retrieved a minimal number of times thereafter, to avoid incurring the extra costs associated with accessing data from the slowest parts of memory. The physical quantity of memory in the system will dictate how careful or how liberal programmers can be when they create data structures. In real-time signal processing, the large quantity of data coming into the system usually requires some amount of buffering in order to ensure that the processors have time to complete their previously assigned tasks. When large amounts of memory are available, higher rate input data streams can be processed by using large buffers to accommodate the latency of the working processors.

The majority of multiprocessor systems have a local memory associated with each processor or processor set. In order to exchange information, especially if the local memory of a processor is insufficient to hold all of the data, a high speed interconnection network is necessary. The bandwidth of the interconnection network as well as that of the input/output paths to the system is a critical hardware feature. Special processing requirements, especially in real-time radar applications, may require the use of third party custom hardware. Some systems are much better suited to foreign boards entering the system backplane. Also, just having extra space in the system cabinet to expand is desirable. Three final hardware criteria that are much more important in mobile platforms are physical size, rugged design, and the system reset time due to either hardware faults or software faults. Finally, there is a dollar cost associated with each system. In the end, project managers must make difficult decisions on whether to absorb the cost for an expensive system up front or to invest the money towards developing cheaper customized hardware that can outperform the general purpose system by specialization. All the considered hardware criteria are summarized below, in no particular order of importance:

- Processing power
- Memory
- Cost
- Size
- Power consumption
- Scalability
- I/O and inter-processor bandwidth
- Ruggedization
- System reset time
- Interoperability with third party hardware

The software development environment is key to unlocking the system's processing potential. Sophisticated compilers that are aware of the underlying hardware can assist in code optimization, efficient and reliable memory use, and portability. On the other hand, a lack of sophisticated compilers may force programmers to learn obscure application programming interface (API) calls or to program in a lower level language, such as assembly language, in order to access hardware features such as the vector processing unit on a processor. Some libraries are usually necessary to lower the programming burden on the programmer, such as standard math routines and message passing libraries to share data between processors. The availability, adherence to standards, and system tuning of these libraries is essential.

Operating systems are important for different reasons during design and the later production phases. During design, an operating system that concurrently handles multiple users writing, compiling, and testing software on a single system is important in order to keep production costs down. It should be noted that during this phase, the availability of diagnostic applications, both for single and multiprocessor routines, is equally important. Fault recovery and isolation is vital during development since programming errors are more likely to appear during this time. Some systems quickly freeze when a single program running on only one of many processors has a fault. This problem severely impacts not only the user who was running the program that crashed, but also every other user on the system. At a minimum, time is wasted waiting for the system to reboot and recover, and then more time is wasted re-opening applications that were being used to develop, debug, and profile the code. At worst, users may permanently lose work that was open when the system crashed.

During the production phase, quick fault recovery is even more important, since for real-time operations, incoming data is either backing up in buffers or being thrown away, and results become stale while the system attempts to recover. System requirements during the production phase move away from accommodating multiple users with their debugging needs to maintaining dedicated use and maximum performance. Many multiprocessor server architectures are severely lacking in this area. Rather than providing a real-time operating system, they take an existing multi-user operating system and tune it to behave like a real-time operating system. One positive aspect of this approach is that the same software that ran during development will more than likely continue to run during dedicated use. Unfortunately, multi-user operating systems tend to be weighed down by interrupts and microcode that can cause unpredictable performance. In addition, precious system memory is lost to running a bloated operating system.

In order to enable future system upgrades or to avoid being tied to a legacy system, software portability has become increasingly important, especially when commercial off-the-shelf hardware is used. Code that achieves high performance across a large number of computing platforms is desirable, but also very difficult to achieve in practice. The Johns Hopkins University Parallel Computing and Imaging Laboratory has made significant strides in developing parameterized algorithms and routines that can be tuned for optimum performance by adjusting parameter values either before or after compilation. This technique greatly reduces the amount of software that must be rewritten when migrating from one hardware platform to another and was applied to the signal processing software developed in this project. Another key to portability is to minimize the use of non-standard libraries and proprietary custom middleware that may need to be rewritten and tuned before the software will run on a new system. The software developed in this project used only native compilers and avoided the use of any proprietary middleware. All software tools utilized were freely available on the host platform.

Some general observations with respect to the above criteria about multiprocessor servers versus embedded systems are made below.

Embedded Processors

- Short reset time
- Small form factor
- Weak compiler optimizations
- Poor software development environment
- Code is not easily portable between different vendor platforms
- Processors tend to lag behind those of servers due to power and thermal constraints
- True real-time operating system support
- Poor fault isolation in a multi-user setting

Servers

- Software easily portable
- Sophisticated tools available for application development, optimization
- Lack real-time operating system
- Large form factor
- Large memory space available

- Excellent multi-user support
- Limited support for third party hardware and less support for custom built hardware

3.0 Aegis Signal Processing Example

The task chosen for this study is a subset of the signal processing performed on the SPY-1B/D radar on Aegis ships. This radar transmits binary coded, phase modulated data. Data processing begins immediately after the A/D conversion of four parallel receiver channels. Initial processing consists of bandpass filtering, coarse doppler compensation, and EMI detection and mitigation. Pulse compression is done to correlate the received signal with the encoded radar signal transmitted. A bank of doppler shifted reference codes is used in the pulse compression operation to provide a finer level of doppler compensation. After all the pulses in a dwell are noncoherently integrated, the target detection process begins. Processing on a dwell concludes with a detection report. All of these steps are shown in the block diagram below.

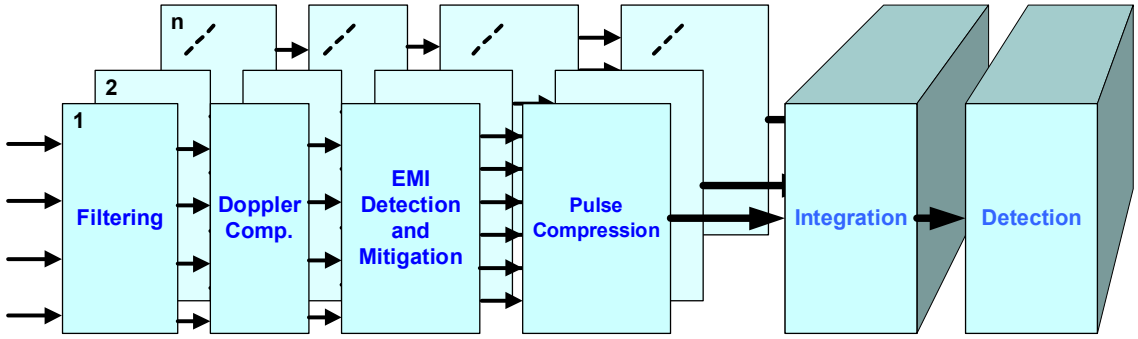

Figure 1: Simplified Processing Flow Diagram

Several software goals have been emphasized in the development process:

- Flexibility Through Parameterization
    - Changes to the specification, such as a requirement to process longer range intervals, should only require adjusting and tuning software parameters, as opposed to either rewriting any code or requiring a new hardware specification
- Efficient Use of Hardware
    - Software should maximize performance on the chosen platform by efficiently utilizing the CPU's memory hierarchy, and interprocessor communication network
- Derive Maximum Benefit From Hardware Upgrades
    - Software should be tunable to take advantage of hardware changes by simply recompiling and running on the upgraded system
- Portability

    - Software should be portable and continue to make efficient use of hardware from different vendors without sacrificing performance so as to maximize flexibility and scope of operation

- Scalability
    - Additional processors can be used to improve performance without rewriting any code

- Validation
    - The software must satisfy the radar performance specifications from an input/output point of view.

Within the framework of the stated goals, we compare the multiprocessor servers and the embedded systems. While the signal processing problem and the methodology of the developers implementing the software have a great impact on satisfying the above software goals, the underlying hardware, operating system, and compilers can either aid or hinder the software team.

Aegis signal processing must satisfy stringent real-time constraints. Two benchmarks crucial for measuring performance are latency and throughput. During real-time operations, the system has a continuous stream of data arriving at the receiver for analysis. The time elapsed from when the last data sample in a dwell leaves the output of the A/D converters to the time a detection report is received at the Radar Control Computer determines the latency of the system. This latency should be on the order of milliseconds. The data arrival rate determines the throughput requirements necessary to ensure that the processing does not lag behind the input data and that no input data is dropped. Latency and throughput are the key metrics used in this study. Latency is measured for a set of simulated input scenarios. Throughput is indirectly measured by determining whether the data for each pulse is processed before the next pulse arrives.

Incorporated in the code are several parameters for tuning performance. The most obvious parameter is the number of processors to use. It should be noted, however, that many contemporary designs use a fixed number of processors and will not run if the number of processors is increased or decreased. Therefore, if a new processor arrives that is twice as fast, and the total system cost can be decreased by using fewer processors, a contemporary design would require a software rewrite, tuning, and test. Other tunable software parameters may be somewhat less obvious. For example, pulse compression is performed using circular convolution. The choice of FFT length in the circular convolution is somewhat arbitrary. In the past, a hardware board that only did a fixed length FFT was incorporated into a typical radar system, and hence the FFT length was not variable. However, multiple operating modes each with different processing demands along with the different memory features of various platforms lead to the observation that a fixed length FFT may not yield optimal performance for all systems and all operating modes. Therefore, FFT length is a software parameter along with the associated radix. For example, a length 1024 FFT may be done as a collection of 256 length-4 FFT's that are themselves combined using a 256 tap FFT, or as a collection of 128 length-8 FFT's that are combined using a 128 tap FFT. Both computations yield the same result, but due to a processor's cache, one may outperform the other. Floating point precision, either 16-bit or 32-bit, is also added as a software parameter to observe the effects of using higher precision for intermediate computation as well as to allow for future systems that may provide greater A/D resolution.

Finally, there are often multiple methods available to pass messages between processors. In a broad sense, there are synchronous and asynchronous send and receive calls. Synchronous communication calls prevent any work from being done while data is being transferred, while asynchronous calls allow computation and communication to proceed simultaneously. Asynchronous communication, however, incurs a small amount of overhead both in programming and execution. If the communication can not be arranged such that significant work is done during the data transfer,

asynchronous communication can actually slow down program execution. Other techniques, such as setting up static buffers for messages may also affect performance. These various message passing techniques are left as software parameters so that the user can best determine what works well on the interconnection network of the system at hand.

4.0 Results

We present results for two operating cases. Four pulses are used in each dwell, however, the quantity of received data that must be analyzed varies widely. Case A is a high resolution search and track mode. In this mode, targets are tracked within a 10 km range gate, and a narrow Doppler passband is used. A Constant False Alarm Rate circuit provides protection in the presence of jamming. Complex data from four channels arrives with 4,400 complex samples per pulse. Case B is a hard limited, low resolution mode with 65,000 complex samples per pulse. This mode is used to cover the entire search volume and as a result, uses a much wider Doppler passband. It should be noted that decimation is performed in Case B, so that the required processing is not a whole order of magnitude greater. The following tables provide latency results using six processors, of which one processor is dedicated to data distribution and another is dedicated to noncoherent integration and target detection. The first table is a for a single pulse dwell. While single pulse dwells are not widely used in practice, the measured latencies indicate how the processing problem might scale if additional processors are added for each pulse. In addition, note that the available Mercury system used had insufficient memory to accommodate multiple pulses in the low resolution mode.

Table 1: Latency comparison for single pulse dwell. Each system is using 6 processors

	HP Superdome	SGI Origin 3800	Mercury (VSIPL)	Mercury (FFTW)
Case A	4.11 ms	7.34 ms	8.86 ms	26.8 ms
Case B	24.20 ms	30.90 ms	TBD	114.7 ms

The above results show the significant benefit obtained from using a vendor tuned library (VSIPL) versus a non-vendor tuned library (FFTW) on the Mercury. The Mercury is only able to obtain good performance when the programmer explicitly makes use of the AltiVec units on each processor. Vendor tuned libraries such as Mercury's Scientific Application Library (SAL) and VSIPL Core Lite make this job easier on the Mercury; however, portability is sacrificed. In contrast, the exact same code can be used on the HP Superdome and the SGI Origin 3800 to obtain a high level of performance.

Table 2: Latency comparison for 4 pulse dwell. Each system is using 6 processors

	HP Superdome	SGI Origin 3800	Mercury
Case A	20.9 ms	40.1 ms	34.6 ms
Case B	101.6 ms	273.6 ms	N/A

The HP Superdome provides the best performance of the systems investigated. Tables 3 and 4 show the minimum size of a server or embedded processor necessary to achieve the required processing throughput. The number of processors given was derived by determining how many multiples of six processors arranged in a round robin fashion would be necessary to accommodate a stream of incoming pulses arriving every 3.125 msec. Six processors are assigned to each incoming pulse to operate on six data channels independently. Figure 2 illustrates how the software scales with the number of processors used.

Table 3: Minimum Server System Size

	HP Superdome	HP V2600	SGI O3800
Case A	12 processors	12 processors	18 processors
Case B	54 processors	90 processors	60 processors

Table 4: Minimum Embedded System Size

	Mercury G4 using VSIPL	Mercury G4 using FFTW
Case A	18 processors	54 processors
Case B	TBD	222 processors

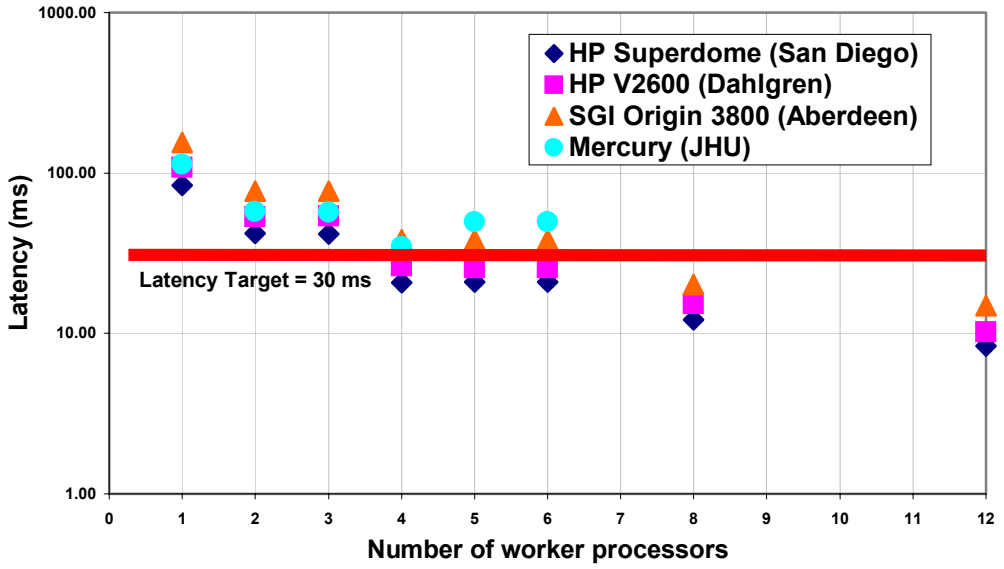

Figure 2. Latency for Single Dwell of 4 Pulses: High Resolution Mode

5.0 Embedded System Notes

There are some obstacles to effectively using embedded systems when compared to multiprocessor systems. Available memory is significantly less per processor. The Mercury system used has only 64MB of memory per processor whereas the multiprocessor servers easily accommodated 512MB and sometimes as much as 2GB per processor. The code that was used in this study would not run for all cases on the Mercury due to the memory layout that was chosen. It should be noted that during early development, a memory layout with tight tolerances was selected, but proved to be limiting when testing different data partitioning schemes. The reason is that depending on the partitioning, a processor is required to allocate memory for a different number of channels. Tracking which processor has which data becomes difficult and can impede performance during the critical junctures where message passing is required.

All of the multiprocessor servers evaluated used superscalar pipelines rather than dedicated vector units. The compilers only have to rearrange code to take advantage of the pipelines, whereas special vector processing calls are needed to use the vector processing units on the Mercury. The easiest way to access the vector units on the Mercury is to use the vendor's proprietary Scientific Application Library (SAL). However, using this library runs counter to the principle of writing portable software, since all of the SAL calls have to be replaced when not running the code on a Mercury computer. A compromise was to use the Vector Signal and Image Processing Library (VSIPL) that is gaining wider acceptance in the signal processing community as a standardized library. A small number of routines were optimized on the Mercury to call SAL with minimal overhead, thereby obtaining the high performance of the vector processing units without complete loss of portability. Incidentally, use of VSIPL on the HP systems increased the execution time because less effort has been made by the vendor to tune the library on these systems.

6.0 Conclusions and Summary

In operational situations where space, power, and system reset time are not critical decision factors, the HP Superdome as well as the SGI Origin 3800 are highly attractive computing platforms because they have powerful compilers, libraries that are fairly general without sacrificing performance, and they have a wide customer base that effectively supports a short product development cycle. The consumers of these machines however are not typically concerned with real-time constraints. Real-time operating system support for these systems lags, a much larger physical volume is required, and a system reset may take several minutes. In an environment where the difference between life and death may be only a few seconds, a long reset time is unacceptable. The result is that while multiprocessor servers can handle many complex problems that require a large amount of memory and many processors, embedded systems have a superior commercial off-the-shelf form factor for radar applications where space is at a premium. However, even when ultimately deploying an embedded system, developing code on a multiprocessor server can be a time saving and useful intermediate step to arriving at the final software for the embedded system.