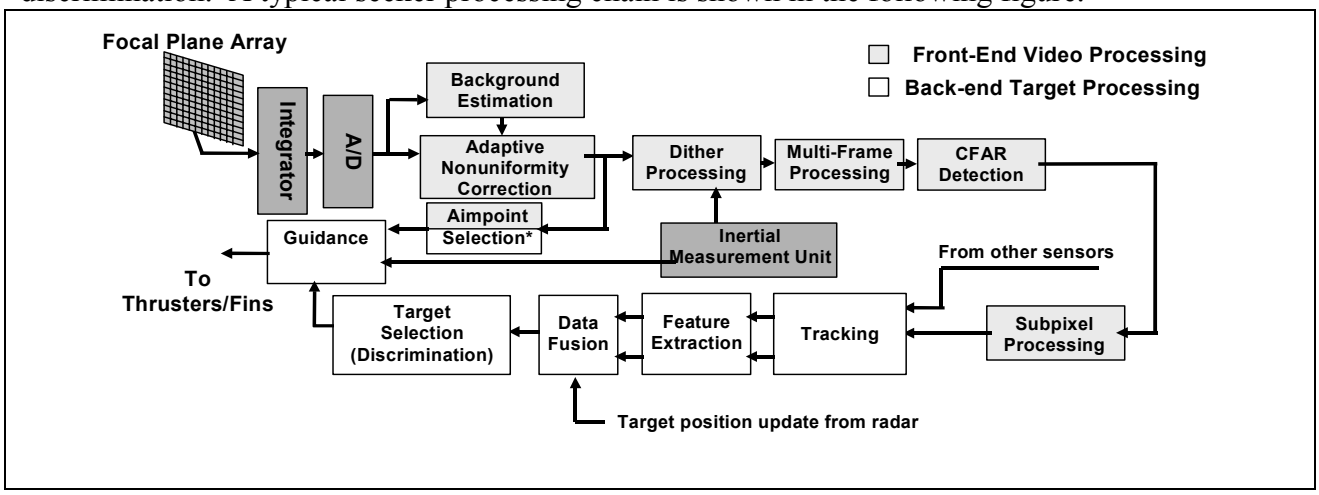
Missile Seeker Common Computer Signal Processing Architecture for Rapid Technology Upgrade*

Daniel V. Rabinkin, Edward M. Rutledge, and Paul Monticciolo
{Rabinkin,Rutledge,Monticciolo}@ll.mit.edu
MIT Lincoln Laboratory, 244 Wood Street, Lexington, MA 02420
June 1, 2002

Interceptor missiles may process IR images to locate an intended target and guide the interceptor towards it. Signal processing requirements have increased as the sensor bandwidth increases and interceptors operate against more sophisticated targets. A typical interceptor signal processing chain is comprised of two parts. Front-end video processing operates on all pixels of the image and performs such operations as non-uniformity correction (NUC), image stabilization, frame integration and detection. Back-end target processing, which tracks and classifies targets detected in the image, performs such algorithms as Kalman tracking, spectral feature extraction and target discrimination. A typical seeker processing chain is shown in the following figure.



In the past, video processing was implemented using application specific integrated circuit (ASIC) components or field programmable gate arrays (FPGAs) because computation requirements exceeded the throughput of general-purpose processors. Target processing was performed using hybrid architectures that included ASICs, digital signal processors (DSPs) and general-purpose processors. The resulting systems tended to be very function-specific, and lack of uniformity and standards resulted in non-portable software. These systems were developed using non-integrated toolsets, and test equipment had to be developed along with the processor platform. The lifespan of the interceptor platform on which a signal processor operates often spans decades, while the specialized nature of processor hardware and software makes it difficult and costly to upgrade. As a result, the signal processing systems often run on outdated technology, signal processing algorithms are difficult to update, and system effectiveness is impaired by the inability to rapidly respond to new threats.

A new design approach is made possible by three developments; Moore's Law - driven improvement in computational throughput; a newly introduced vector computing capability in

* This work is sponsored by the United States Navy Standard Missile Program PMS-422. Work performed by MIT Lincoln Laboratory is covered under Air Force Contract F19628-00-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Air Force or the United States Navy.

general purpose processors; and a modern set of open interface software standards. Today's multiprocessor commercial-off-the-shelf (COTS) platforms have sufficient throughput to support interceptor signal processing requirements. An application may be programmed under existing real-time operating systems using parallel processing software libraries and middleware, resulting in highly portable code that can be rapidly migrated to new platforms as processor technology evolves. Use of standardized development tools and 3rd party software upgrades are enabled. Rapid processor technology upgrades and algorithms updates can be implemented. The resulting weapon system will have a superior processing capability over a custom approach at the time of deployment as a result of a shorter development cycles and use of newer technology. The signal processing computer may be easily upgraded over the lifecycle of the weapon system, and may easily migrate between weapon system variants because of modification simplicity.

We present a reference design using the new approach that utilizes an Altivec PowerPC parallel COTS platform with 4 processors. It uses a VxWorks-based real-time operating system (RTOS), and application code developed using an efficient parallel vector library (PVL) that provides computational functionality and C++. PVL abstracts the parallelization of application code by introducing maps and grids associated with data structures and operations that define how data and processing are partitioned across a parallel processing platform. To port the application, only the maps and grids are modified – the application code remains unchanged.

The seeker application was developed in several stages. Initially, the algorithm blocks were developed and debugged using MATLAB. The code was converted to PVL/C++ and implemented on a workstation. The processing results were verified against MATLAB results. The PVL code was then migrated to a PowerPC based network of workstations to verify proper parallel operation. Finally the code was ported to a real-time embedded platform to demonstrate real-time operation. The staged approach enabled development in a high-level environment with minimal effort required for the consequent move to a parallel platform. The process of migrating applications from PVL running on a single workstation to PVL running on the parallel embedded real-time platform proceeded very rapidly, with over 99% code portability achieved at the application level. The lines of code (LOC) counts for the front-end processing components are given in the table below.

Component	Matlab LOC	PVL/C++ LOC	Matlab-to-C Compiler LOC
ADNUC	36	143	538
Dither Subtraction	28	160	437
Integration	11	63	178
CFAR	2	60	90
Bulk Filter	5	152	211
Total	82	578	1454

Observe that applications require roughly 6 PVL/C++ lines for every line of MATLAB. This may be compared with results from the MATLAB-to-C compiler for single-processor code. Additional overhead needed to run such code on a parallel platform was estimated to roughly double LOC. This indicates that PVL/C++ provides rough a 5:1 reduction in LOC relative to code that does not utilize middleware. Real-time operation was achieved 3 months after the initial port, and a speedup of >2 was obtained by running the code on 3 of the processors. The system was tested on simulated and real sensor data, and proper processing was verified.