WHITE PAPER

# Xeon-D Vs Xeon-E for embedded radar applications

## Comparing server-class devices for Space Time Adaptive Processing applications

### Introduction

For airborne radars, longer, higher and further means more processing power in smaller, rugged efficient packages. The best implementations of ANSI/VITA 65 (OpenVPX), the de facto embedded military open system compute architecture meets the ruggedness and compact requirements, and adds scalability. Especially when implemented as VPX-REDI (VITA 48) even greater ruggedness and increased functional density is possible, as is the ease of two-level maintenance (2LM). Processing power is achieved by leveraging the best commercial Intel Xeon data-center compute capability. Xeon processors are available as either "mobile" devices (Xeon D) which are designed for laptop applications which require lower power, less cores, lower memory bandwidth and less connectivity. Xeon E processors power data-centers and cloud facilities worldwide. Such processors typically have larger core counts, faster memory and increased connectivity such as QPI enabling efficient use of multiple on-board processor and SMP.

Embedding Xeon E devices in to military applications requires rugged packaging, reliable/efficient cooling, fast and unrestricted pipes and banks of memory. Mercury's proven (fourth generation) OpenVPX Xeon E powered blades have these enabling technologies and are known as the Ensemble® HDS (High Density Server) series of blades., How do these embedded blades with data-center performance usher in the next generation of radar systems? In the past Space Time Adaptive Processing (STAP) was a challenge for embedded systems. This white paper studies STAP processing approaches, using Xeon D and E processors for comparison.

JONAS LARSSON
PRINCIPAL SYSTEMS APPLICATION ENGINEER
APRIL 2017

mercury systems™

## Space Time Adaptive Processing (STAP)

STAP radar systems adaptively compute weights to reduce the effect of clutter and jamming. Targets in motion relative to the radar will present a Doppler shift in the returned radar echo. In typical moving target indicator (MTI) radars, this is taken advantage of. However, if the radar platform is in motion, such as in airborne radar systems, then also the ground will present a Doppler shift. In such environments it can be challenging to separate ground clutter from targets. Fortunately, such ground clutter typically provides a similar Doppler shift for the area adjacent to the area being examined. By constructing a filter which will take such an adjacent area into consideration, a STAP system can reduce the effect of ground clutter. With the use of an antenna array and adaptive weights, the STAP system can also adapt the radar antenna pattern by placing nulls in the direction of jammers.

Computing the adaptive weights in real-time is an intensive process; computational burden can be reduced by selecting computations most suitable for the processing technology. Even if the most suitable approach is selected for a high-performance multi-channels STAP system, the processing demand can be substantial. As such, previous generation compute solutions were unable to meet size, weight and power (SWAP) requirements, making STAP difficult to deploy.

Next generation processor, GPU and FPGA technology enables the deployment of effective STAP systems in airborne platforms. While multiple technologies can be used, the technology can greatly affect the programmability, scalability and life-cycle management of the platform.

This white paper analyses Xeon server-class processors in airborne STAP systems. It is not an evaluation of current techniques in STAP science and practice. Instead it is an evaluation of new compute technologies that target this class of processing, while comparing mobile (D) and data-center (E) Xeon processing capabilities.

## STAP algorithm

Figure 1 illustrates the STAP processing blocks. This STAP algorithm is further described by Cain et al.

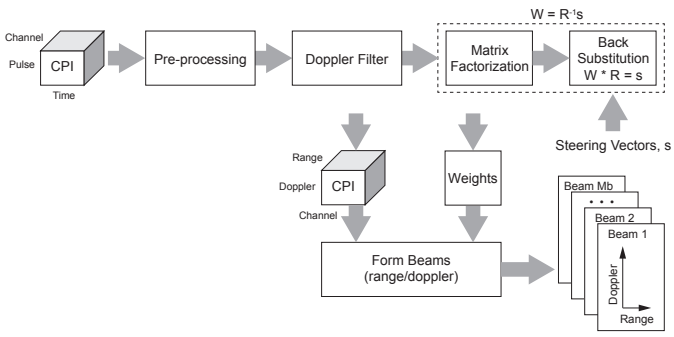


***Figure 1. STAP overview***

The initial stage with pre-processing typically includes Video-to-IQ conversion, array calibration and pulse compression followed by Doppler filtering. The Video-to-IQ conversion usually includes demodulation to baseband, low-pass filtering and decimation of sample rate. These processing blocks are illustrated in Figure 2.

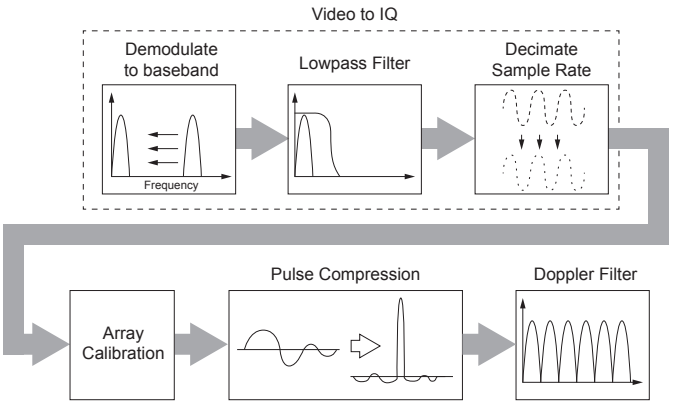


***Figure 2. Pre-processing***

The parameters used in this section are introduced in Table 1.

***Table 1. Pre-processing parameters***

| Pre-processing parameters | |
|---|---|
| Nr channels | L |
| Nr pulses per CPI | $P_{CPI}$ |
| Nr pulses per Doppler processing block | $P_D$ |
| Samples per pulse before decimation | N |
| Decimation factor | D |
| Samples per pulse after decimation | $N_D$ |
| FIR filter length used anti-aliasing in video-to-I/Q | $K_a$ |
| FFT size (power of 2) used by overlap-save fast convolution | R |
| Convolution length in calibration and pulse compression | $R_{cp}$ |
| Number of blocks in the overlap-save fast convolution method B = ND / Rcp | B |
| Doppler FFT Size (power of 2) | K |

The number of operations required for the pre-processing is listed in Table 2

***Table 2. Pre-processing number of operations***

| Nr Operations | | |
|---|---|---|
| Video-to-IQ | Demodulation to baseband | $L * P_{CPI} * 2N$ |
| | Low-pass filter (anti-aliasing using FIR filter) and decimate sample rate | $L * P_{CPI} * 3K_a * N_D$ |
| Array calibration & pulse compression | Array calibration and pulse compression total | $L * P_{CPI} * B * (10R * \log_2 R + 6R)$ |
| | Forward FFT to get into frequency domain | $L * P_{CPI} * B * (5R * \log_2 R)$ |
| | Multiplications in frequency domain | $L * P_{CPI} * B * 6R$ |
| | Inverse FFT to return to time domain | $L * P_{CPI} * B * (5R * \log_2 R)$ |
| Doppler processing | | $L * N_D * (5K * \log_2 K + 2P_D)$ |

## Adaptive weight processing

The most demanding processing stage in STAP is to calculate and apply the adaptive weights vectors. The weight vectors are then used to form beams in beam forming. This process is described in Figure 3.

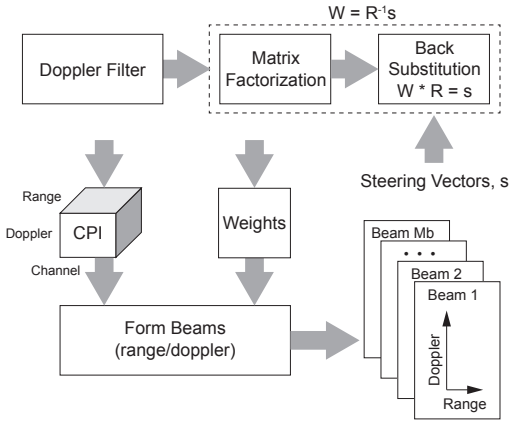


***Figure 3. Adaptive processing***

Calculating such adaptive weights may be performed in several ways. The two most commonly used methods to solve this equation in STAP applications are Cholesky and QR decomposition.

## Cholesky method

With the Cholesky matrix method we first solve the following equation,

$M_{cov} * V_{weight} = V_{steer}$

We define $M_{cov} = Z * Z^H$ and compute $M_{cov}$

Solving system $V_{weight} = M_{cov}^{-1} * V_{steer}$ determines the upper triangular R. We then use forward and backward substitution to compute weight vector.

In the Cholesky approach, the data samples (i.e. voltages) are multiplied together to form the elements of the covariance matrix. The units of these elements are 'power', and thus the numerical dynamic range of the elements (in dB) are double that of the elements in the original data. As a result, especially in the presence of interference signals having a high interference-to-noise ratio, the resulting covariance matrix can rapidly become ill-conditioned requiring double-precision arithmetic to obtain an accurate decomposition. The QR decomposition approach described in the following section avoids this problem by working directly with the data in the 'voltage' domain, rather than with the 'power' elements of the covariance matrix.

## QR decomposition method

With the QR decomposition method you would apply the QR decomposition directly to the complex data matrix to obtain the desired upper triangular factor matrix R. In this method we decompose space-time matrix MxN for some range cells. We then use forward and backward substitution to compute weight vector. Once we have the weight vector we apply it using vector-matrix multiplication.

A drawback with the QR decomposition method is that it requires nearly twice as much computation as Cholesky but given the finite-precision arithmetic inherent in all digital processors, the QR decomposition method is numerically more stable than the Cholesky method.

Using the QR decomposition method the number of operations required for weights computation and application is described in Table 3. The parameters used are K Doppler FFT size (power of 2), M independent non-overlapping range blocks, L channels, Q processing order, and $N_R$ contiguous range cells per weight computation.

***Table 3. Adaptive processing***

| Functional Block | Functional Block | Nr Operations |
|---|---|---|
| Adaptive processing | Weights computation | $K * M * 8 * [L*Q]^2*(N_R+1)$ |
| | Weights application | $K * M * (8 * L* Q * N_R)$ |

A first-order Doppler-factored STAP (Q = 1) represents a first-order post-Doppler adaptive DPCA algorithm. This is a basic post-Doppler STAP for clutter and interference suppression. It can be effective but since it only operates in a single temporal degree of freedom (DOF) it is not a true STAP. The processing demand for this algorithm is modest.

A third-order Doppler-factored STAP provides performance approaching a fully adaptive system. This approach efficiently supresses clutter and interference. The processing demand for this algorithm can be high, especially for a high channel count. This is where we will focus this study.

## STAP parameters and processing requirements

In order to better understand the computational load related to these algorithms this section analyses the effect of some of these. For this exercise we have assumed the parameters listed in Table 4.

***Table 4. STAP Parameters***

| Parameter | Name | Value |
|---|---|---|
| IF sampling rate [MHz] | $F_s$ | 5 |
| Time per pulse, $T_P = T_{CPI} / P_{CPI}$ [us] | $T_P$ | 504 |
| Time per CPI $T_{CPI}$ = 32.25 ms [ms] | $T_{CPI}$ | 32.25 |
| Range = C / (2*PRF) [km] | Range | 75 |
| Pulses per second, PRF = $1/T_P$ [kHz] | PRF | 1984 |
| Samples per CPI, $N_{CPI} = P_{CPI} * N$ | $N_{CPI}$ | 122,880 |
| Nr channels | L | 22 |
| Nr pulses per CPI | $P_{CPI}$ | 64 |
| Nr pulses per Doppler processing block | $P_D$ | 64 |
| Samples per pulse before decimation | N | 1920 |
| Decimation factor | D | 4 |
| Samples per pulse after decimation | $N_D$ | 480 |
| FIR filter length used in video-to-I/Q | $K_a$ | 36 |
| FIR filter length used in array calibration | $K_c$ | 3 |
| FIR filter length used in pulse compression | $K_p$ | 63 |
| Convolution length in calibration and pulse compression | $R_{cp}$ | 192 |
| FFT size (power of 2) used by overlap-save fast convolution | R | 256 |
| Number of blocks in the overlap-save fast convolution method $B = N_D / R_{cp}$ | B | 3 |
| Doppler FFT Size (power of 2) | K | 64 |
| Number of independent non-overlapping range blocks | M | 2 |
| Nr contiguous range cells per weight computation | $N_R$ | 240 |
| Processing order | Q | 3 |

These are similar to the parameters used by Cain et al. [1]. As shown in previous sections the amount of processing required to perform STAP in real-time is highly affected by parameters such as sampling speed and Degree of Freedom (DOF). Since this study focus on a third-order Doppler-factored STAP with 22 channels (L) and processor order (Q adjacent Doppler bins) set to 3 we have a degree of freedom (DOF=L*Q) of 66. As previously discussed the number of operations required to adaptively compute weights increases with $DOF^2$ which means that a doubling of either channels (L) or processing order (Q) will quadruple the operation rate.

The following figures illustrate the effect of the number of channels on the various processing blocks. Here we can see that weight computation dramatically increases with the number of channels to process.

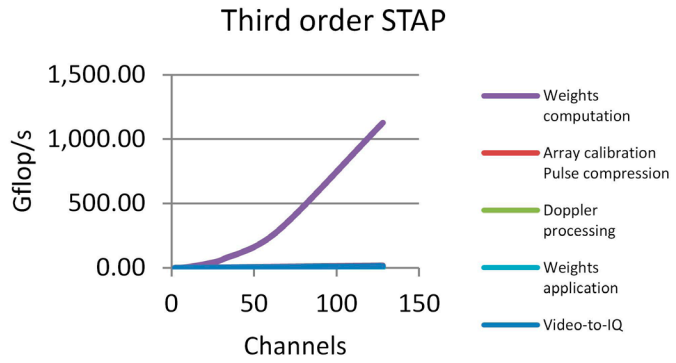


***Figure 4. Processing versus channels***

As illustrated in the Figure 4, for STAP processing it is typical that weight computation forms a majority of the overall processing requirements. Therefore in order to assess the usability of a particular platform it seems logical to focus a STAP feasibility study on the efficiency to perform weight computation.

If we perform weight computation using QRD method then this involves decomposition of space-time matric MxN for some range cells (e.g. 240). We then use QR decomposition to give the upper triangular R. Finally, we use forward and backward substitution to compute the weight vector.

Depending on the optimised implementation the data might need to be rearranged to suite the underlying architecture. For instance an optimised Gram-Schmidt QR decomposition requires the transpose of the matrix.

## Benchmark development

The STAP benchmark used herein is based on the MITRE RT_STAP benchmark [1]. The benchmark was originally developed twenty years ago, to run on processor technology available at the time of the benchmark. Since then, technology has moved forward. Despite evolving hardware all the involved APIs such as math library (Mercury MathPack) and data movement (MPI) has been maintained during this time. As a result of this the porting activity has mainly involved setting up the build environment and to build against the latest libraries for the selected processor type.

## Optimisations

### Function mapping file

Using a function-by-function selection mapping file the most efficient function was selected from multiple optimized math libraries (e.g. Intel IPP, Intel MKL and Mercury MathPack/SAL).

### Low-pass FIR filter

The low-pass filter (FIR) function was further optimized to use the Intel AVX2 FMA (fused-multiply-add) and permute operations.

### Dot product

The dot product function was also optimized to use AVX2 FMA. This improved QRD and weight application.

## Test system

As mentioned in the introduction, STAP could be implemented using FPGAs, GPUs or CPUs. This paper discusses STAP on CPUs and specifically Intel server-class processors.

The measurement presented here is based on a high-performance Haswell dual-server Intel E5-2648L v3 running at 1.8 GHz (used on Mercury OpenVPX board Ensemble HDS6603). The memory consists of four banks of DDR4-2133 per processor (eight in total) with a total of 136 GB/s. The memory speed is far beyond a typical embedded system and allows efficient usage of the many cores, for both compute bound and memory bound applications.

Two 12-core processors are interconnected with dual QPI with a total of 38 GB/s bi-directional (76 GB/s total). With the dual QPI interconnecting both processors which runs in SMP they appear from a software development as well as performance point of view as a single 24-core processor.

For external communication the processors share six x8 PCIe3 operating at 8G resulting in 48 GB/s bi-directional (96 GB/s total) off-board data links. Two third of this (four x8 PCIe3) are dedicated to data input interfaces and other adjacent equipment such as recorder. Having separate data paths for data input/output, inter-board links and on-board links between processors further ensures the ability to keep the 24 cores occupied at all times.

## Measured performance

The benchmark was run and the generated data was successfully validated. This section describes the achieved performance on the Xeon E (Intel E5-2648L v3).

### Low pass filter (FIR)

Following previously described optimizations the achieved performance for FIR filter reached 25.5 Gflop/s per core. When combined with decimation and other miscellaneous house-keeping code, the higher level function performance is reduced somewhat. This is clarified in the following sections.

### FFT

In-place Interleaved Complex Fourier Transform (FFT) performance has been measured using Intel IPP and MKL libraries. The performance is shown in Figure 5. The data was placed in cache before timing.

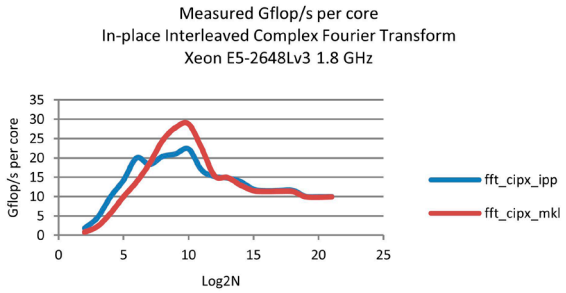


***Figure 5. In-place Interleaved Complex Fourier Transform (FFT) performance***

The most suitable library depends on the FFT size. For in-place interleaved complex Fourier transform, Intel MKL was used.

### QR-Decomposition

Using the QR decomposition benchmark described in MITRE RT_STAP [1] the following timing per core for various QRD sizes have been measured.

***Table 5. QR decomposition performance***

| Matrix size | Avg processing time (msec) | Gflops/s |
|---|---|---|
| 32x16 | 0.008 | 8.1 |
| 64x32 | 0.037 | 14.3 |
| 96x48 | 0.105 | 16.9 |
| 128x64 | 0.233 | 18.0 |
| 160x80 | 0.560 | 14.6 |
| 192x96 | 0.946 | 15.0 |
| 64x16 | 0.011 | 11.4 |
| 128x32 | 0.072 | 14.5 |
| 192x48 | 0.180 | 19.7 |
| 256x64 | 0.522 | 16.1 |
| 320x80 | 1.026 | 16.0 |
| 384x96 | 1.786 | 15.9 |

The code used during testing is based on a loop of discrete calls to lower level functions such as a vector conjugate inner dot product function cidotprx. This is a relatively low level function operating at separate vectors and it is optimized. The usage of optimized functions explains the relatively high performance. In order to further improve the advantage of using the server-class architecture a more monolithic function operating directly on the matrix, ideally for the complete QRD might provide further improvements.

### Third-order Doppler-factored STAP – single core

The third-order Doppler-factored STAP is an application level STAP benchmark involving 22 channels. The result of running this on a single core is presented in Table 6.

Weights computation (QRD) clearly requires the most computations (83.4%) and even with the use of an optimized library the core spends 66.5% of its time here.

With a single core, the total processing time of 102.2 milliseconds is longer than the 32.5 millisecond per CPI. To perform the STAP calculations before the next CPI arrives, add more cores and perform calculations in parallel (next section).

The listed total operation rate of 12.57 Gflop/s is somewhat misleading. It is calculated as total run time / Nr float operations per CPI. The total run time includes other processing such as short to float cast, misc. preprocessing, misc. processing and corner-turn. The operations per CPI for such processing have not been accounted for (only its time). This means that the actual overall operation rate is somewhat higher.

### Third-order Doppler-factored STAP – four cores

If we run the third-order Doppler-factored STAP benchmark with four cores working in parallel we reach below the CPI time and thereby meet the benchmark requirements. The measured timing for running with four cores in parallel (using MPI) is shown in Table 7.

***Table 6. Third-order Doppler-factored STAP - single core***

| Functional block | Function | Nr float operations per CPI | Nr float operations of total [%] | Operation rate required [Gflop/s] | Single core measured time [s] | % of total time | Single core operation rate [Gflop/s] |
|---|---|---|---|---|---|---|---|
| Pre-processing | Short to float cast | | | | 0.0031 | 3.0% | |
| | Demodulation to baseband | 5,406,720 | 0.4% | 0.17 | 0.002 | 2.0% | 2.71 |
| | Low-pass filter (FIR) and decimation | 72,990,720 | 5.7% | 2.26 | 0.00393 | 3.8% | 18.58 |
| | Pulse compression array calibration | 92,995,584 | 7.2% | 2.88 | 0.00296 | 2.9% | 31.42 |
| | Misc. pre-processing | | | | 0.0006 | 0.6% | |
| Adaptive processing | Doppler processing | 21,626,880 | 1.7% | 0.67 | 0.00075 | 0.7% | 28.84 |
| | QRD | 1,070,530,560 | 83.4% | 33.19 | 0.06797 | 66.5% | 15.76 |
| | Solve for adaptive weights | 4,460,544 | 0.3% | 0.14 | 0.00388 | 3.8% | 1.15 |
| | Weights application | 16,220,160 | 1.3% | 0.50 | 0.00171 | 1.7% | 9.49 |
| Other | Corner-turn 2x | | | | 0.00455 | 4.5% | |
| | Misc. processing | | | | 0.01077 | 10.5% | |
| | Total run time | 1,284,231,168 | 100% | 39.82 | 0.10221 | 100.0% | 12.57 |

*Table 7. Third-order factored-Doppler STAP - four cores*

| Functional block | Function | Nr float operations per CPI | Nr float operations of total [%] | Operation rate required [Gflop/s] | Four cores measured time [s] | % of total time | Four cores operation rate [Gflop/s] |
|---|---|---|---|---|---|---|---|
| Pre-processing | Short to float cast | | | | 0.00085 | 3.0% | |
| | Demodulation to baseband | 5,406,720 | 0.4% | 0.17 | 0.00055 | 1.9% | 9.84 |
| | Low-pass filter (FIR) and decimation | 72,990,720 | 5.7% | 2.26 | 0.00105 | 3.7% | 69.52 |
| | Pulse compression array calibration | 92,995,584 | 7.2% | 2.88 | 0.00085 | 3.0% | 109.41 |
| | Misc. pre-processing | | | | 0.00015 | 0.5% | 0 |
| Adaptive processing | Doppler processing | 21,626,880 | 1.7% | 0.67 | 0.0002 | 0.7% | 108.14 |
| | QRD | 1,070,530,560 | 83.4% | 33.19 | 0.01731 | 61.2% | 61.85 |
| | Solve for adaptive weights | 4,460,544 | 0.3% | 0.14 | 0.00097 | 3.4% | 4.6 |
| | Weights application | 16,220,160 | 1.3% | 0.50 | 0.00041 | 1.4% | 39.57 |
| Other | Corner-turn 2x | | | | 0.00314 | 11.1% | |
| | Misc. processing | | | | 0.0028 | 9.9% | |
| | Total run time | 1,284,231,168 | 100% | 39.82 | 0.02829 | 100.0% | 45.4 |

As per Table 7 the measured time to run the STAP benchmark on four cores is 28.29 ms which is below the CPI time of 32.5 ms. Utilizing four cores provides an overall 45.4 Gflop/s which is also above the required 39.82 Gflop/s.

## Measured scalability

Using MPI the STAP benchmark has been scaled to run on varying number of cores. The measured aggregate operation rate is shown in Figure 6.

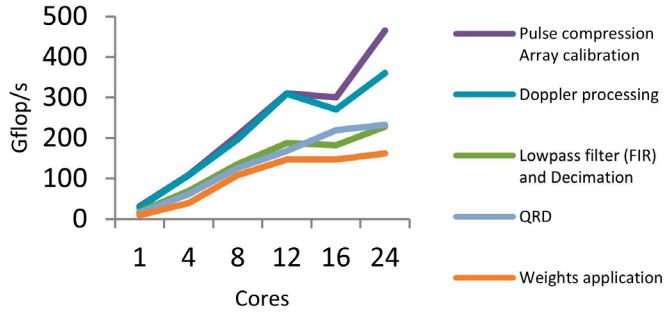


***Figure 6. Measured scalability***

The measured operation rate per core is more clearly illustrated in Figure 7.

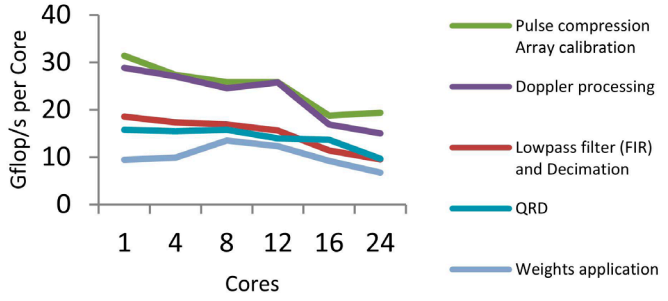


***Figure 7. Measured scalability Gflops/s per core***

Most parts of the application have a reasonable scaling, some scale factors are more efficient than others which are application and data size specific. Due to latency induced by data movement, using also the 2nd processor (cores 13-24) reduces the per-core performance. The dual QPI interlinking the processors helps reducing such effect. Since the problem size used herein can be solved using four cores, scaling this to 24 cores (as shown in this section) is more academic than of practical use. A more demanding application (e.g. with larger data sizes) would scale more evenly over multiple processors. Instead of splitting a single QRD over many cores it would be more efficient to run multiple QRDs in parallel on groups of cores. A STAP application might be partitioned differently from this benchmark resulting in improved scalability.

The level of scalability depends on the type of computation and partition. As shown below, for compute bound problems (small FFT sizes) the scalability of algorithms is straight-forward but for larger FFT sizes the problem becomes memory bound (data does not fit in cache). This is shown in Figure 8.

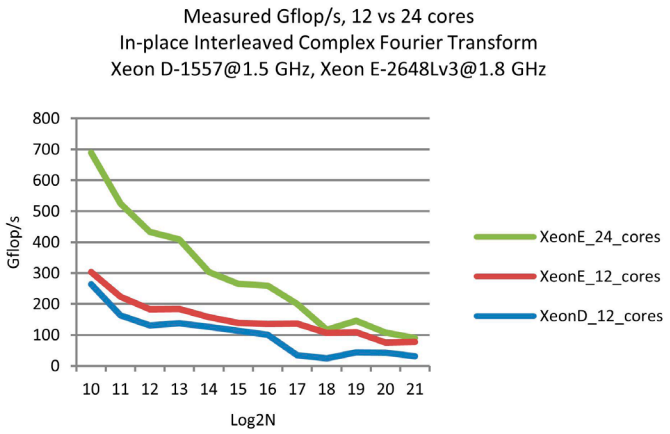


***Figure 8. Measured Gflop/s 12 Vs 24 cores***

Figure 8 shows both Xeon D and E performance. The difference between these is discussed in the next section.

## Xeon D Vs E

As of today there are embedded products available with dual server-class Xeon D-1540/1557 and Intel Xeon E5-2648L v3. An illustration of a dual Xeon D versus dual Xeon E architecture is shown in Figure 9

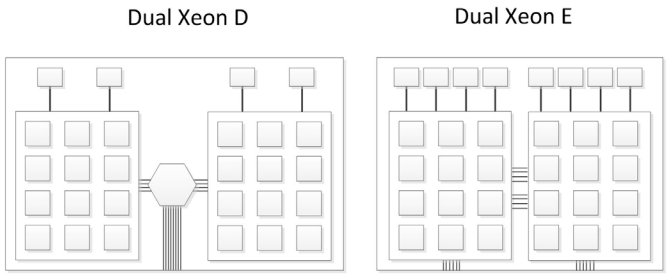


***Figure 9. Dual Xeon D and E architecture comparison***

Figure 9 has two Xeon processors on each board. Both these processors have AVX2 arithmetic units.

The Xeon E design is derived from a multi-socket high-performance server computer. With its double memory bandwidth and high-speed processor interlink it is clearly designed for higher performance applications. It can also run in SMP mode (one common OS running on both CPUs) with all 24 cores sharing a common high-speed memory.

The eight-core Xeon D-1540 has 12 MB cache and the twelve-core Xeon D-1557 has 18 MB cache. The Xeon E-2648Lv3 with 12 cores has 30 MB cache. The cache size is critical for many applications so this is a major difference. A larger cache allows a processor to work on larger data sets and cache size can be even more important when more cores share a common cache. The Xeon D has two memory channels per processor and the Xeon E has four. The result is that the Xeon D memory speed is half of Xeon E. The effect of the difference in cache size and memory speed is illustrated in the Figure 10 (FFT with Intel IPP).

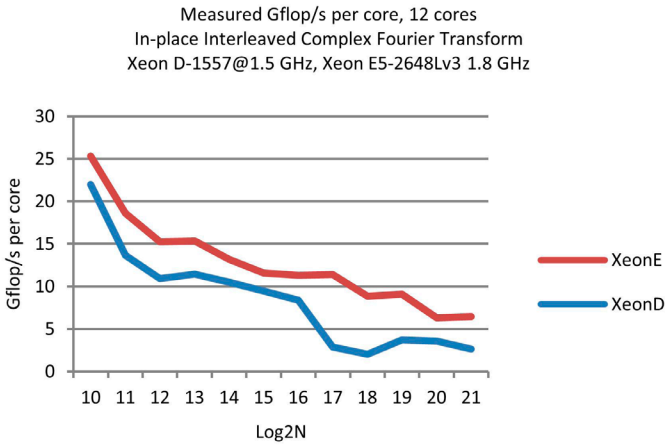


***Figure 10. In-place interleaved complex FFT***

If the Xeon D was run at the same clock speed of 1.8 GHz as the Xeon E (E5-2648Lv3) and running small FFT size, the performance per core would be similar. As the performance for larger sizes is limited by the cache size and memory bandwidth (memory bound), for larger FFT sizes, using more cores or increase core clock speed would not improve performance.

In order to further clarify this, the diagram below shows the performance for scenarios where we run FFT on 6 and 12 cores simultaneously, all competing for the same cache and memory resources.

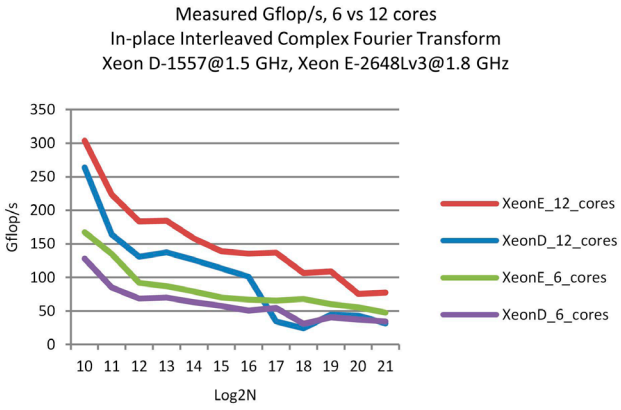


***Figure 11. Measured Gflop/s, 6 Vs 12 cores***

As shown in Figure 11, for memory bound functions, six Xeon E cores can outperform twelve Xeon D cores.

If we run the full STAP benchmark on both Xeon D and E we will see that up to four cores, Xeon D can keep up with Xeon E. This is shown in Figure 12.

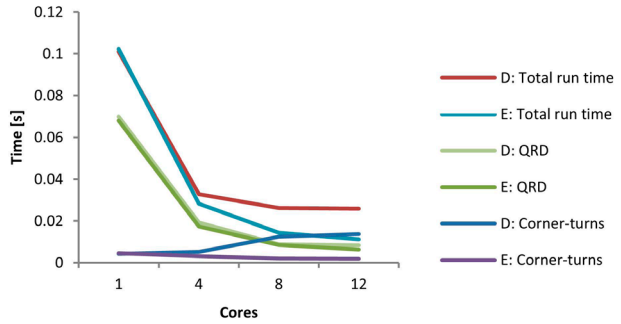


***Figure 12. STAP timing Xeon D Vs E***

While compute bound functions run in isolation they perform similarly but the cost of partitioning the full benchmark across cores and the resulting distributed corner-turns result in the Xeon D struggling. As a result, when using all twelve cores, it takes more than twice the time to run this benchmark on Xeon D. However, it seems likely that the original software can be redesigned to improve scalability for both Xeon D and E.

The Xeon D's lack of a direct high-speed interlink between the processors (QPI) results in a scalability between on-board processors to be the same as scaling across boards (over the fabric). On a single board the dual Xeon D are interconnected with 16x PCIe3 at 8 Gbps resulting in 16 GB/s compared with Xeon E's dual QPI with 38 GB/s BW. The result is that moving data between on-board processors will take twice the time.

In applications demanding higher memory speed and inter-processor communication the Xeon E's larger cache, double memory speed and inter-processor QPI links will come into even greater importance.

Such applications might be ground based 100+ channel radar systems with higher pulse repetition frequency (PRF) implementing an ordered statistics CFAR-algorithm operating in all three dimensions (range, pulse and channel). In such system, whilst computation complexity could be somewhat reduced, data movement is more demanding.

Another application where memory speed and data movement is important is higher resolution Synthetic Aperture Radar (SAR).

The analysis of using server-class processors in such applications could be a subject for further studies.

## Summary and conclusions

QR-Decomposition requires a majority of the processing in STAP radar. As shown in this paper reaching in the order of 15 Gflop/s per core for QRD is something we can expect from an Intel server computer running optimized math libraries. It should also be possible to optimize this further.

STAP radar will however need to do more than QRD and therefore we have studied the performance of running a somewhat more complete STAP application based on MITRE RT_STAP [1]. Given the presented data it appears realistic to expect at least 10 Gflop/s per core overall STAP computations reaching to 30 GFlop/s per core for some functions (e.g. FFT).

With the used STAP example application we would need about four cores to process the incoming data stream of 22 channels in real-time. One may ask how many channels can be processed in a slot or even in a 19" rack. However since real world sampling speed can be higher and other factors come into play (e.g. increased processing order, degree of freedom, etc.) making such general statement is not practical.

What we can envision is how much STAP processing can be done in a compact 6U embedded chassis. With OpenVPX technology it is practical to build a compact rugged deployed embedded system in the region of 10 slots. Typically in a deployed system there will be some slots for data input. There will also be a switch card as well as need for spare slots. A typical deployed system might therefore have around five server processor boards. If each board has 24 cores then we would have a 120-core STAP computer. Assuming we reach between 10-20 Gflop/s per core overall then this would allow us to reach at least 1.2 Tflops/s and possibly extending up to 2.4 Tflops/s of STAP processing per 19" rack.

We have compared Xeon D Vs E and measured that for the same core count and clock speed, compute bound functions run in isolation perform similarly. But even with a compute bound application such as STAP there will be data movement such as corner-turn which takes time. Xeon E has twice the memory speed which reduces time for memory data movement by up to 50%. It also has nearly double the cache size which to a greater extent enables the cores to work from cache instead of memory. As shown herein this makes scaling across cores more straight-forward. The effect of cache size and memory speed can also be greatly enhanced in memory-bound applications requiring more data movement. The higher performance of Xeon E allows us to reduce the number of boards and this reduces system size, weight (lower-SWaP) and complexity.

As described herein commercial embedded server-class processing technology is ready to make 3rd order STAP and beyond a reality in deployed systems.

## Table of Acronyms

| | |
|---|---|
| CPU | Central Processor Unit |
| DOF | Direction Of Freedom |
| FFT | Fast Fourier Transformation |
| FMA | Fused Multiply Add (Intel) |
| FPGA | Field Programmable Gate Array |
| GPU | Graphical Processing Unit |
| MTI | Moving Target Indicator |
| OS | Operating System |
| QPI | Quick Path Interconnect (Intel) |
| SMP | Symmetric Multi-Processing |
| STAP | Space Time Adaptive Processing |

## References

[1] K.C. Cain, J. A. Torres, and R.T. Williams, "RT_STAP: Real-time space-time adaptive processing benchmark", MITRE Technical Report, The MITRE Corporation, Center for Air Force C3 Systems, Bedford, MA, USA, 1997.

## About the Author

**Jonas Larsson** is a Principal Systems Application engineer for Mercury Systems. Mr Larsson has 18 years of experience architecting, implementing and promoting high-performance embedded system solutions. Mr Larsson earned his bachelor of science (BSc) degree in electrical engineering from Chalmers University of technology in Sweden and his master of science (MSc) degree in Network centred computing from University of Reading in England.

 3314.00E-0417-wp-STAP



mercury systems™

*INNOVATION THAT MATTERS™*

**Corporate Headquarters**
50 Minuteman Road • Andover, MA 01810 USA
(978) 967-1401 • (866) 627-6951 • Fax (978) 256-3599

**Europe - Mercury Systems, LTD**
Unit 1 - Easter Park, Benyon Road, Silchester, Reading
RG7 2PQ United Kingdom
+ 44 0 1189 702050 • Fax + 44 0 1189 702321