JetBench: An Open Source Real-time Multiprocessor Benchmark

Muhammad Yasir Qadri, Dorian Matichard, and Klaus D. McDonald Maier

School of Computer Science and Electronic Engineering
University of Essex, CO4 3SQ, UK[1, 3]
Ecole Nationale d'Electronique, Informatique et Radiocommunications de Bordeaux, ENSEIRB[2]
yasirqadri@acm.org[1], matichar@enseirb.fr[2], kdm@essex.ac.uk[3]

Abstract. Performance comparison among various architectures is generally attained by using standard benchmark tools. This paper presents JetBench, an Open Source OpenMP based multicore benchmark application that could be used to analyse real time performance of a specific target platform. The application is designed to be platform independent by avoiding target specific libraries and hardware counters and timers. JetBench uses jet engine parameters and thermodynamic equations presented in the NASA's EngineSim program, and emulates a real-time jet engine performance calculator. The user is allowed to determine a flight profile with timing constraints, and adjust the number of threads. This paper discusses the structure of the application, thread distribution and its scalability on a custom symmetric multicore platform based on a cycle accurate full system simulator.

Keywords: Real-time, Multiprocessor, Application Benchmark

1 Introduction

Benchmarks are generally classified into two types, i.e. 1) synthetic benchmarks and 2) application benchmarks. Synthetic benchmarks are designed to exploit particular property of a processor such as instruction per second (IPS), cache performance, I/O bandwidth etc, whereas application benchmarks are centred towards one particular application such as automotive, office automation, etc. The concept of using benchmarks for performance characterization of the system is common practice and some processor manufacturers have proposed their own benchmarks [1]. However such benchmarks strive to give better performance on a particular platform, third party benchmarks are a good way to compare the performance amongst various architectures impartially and transparently.

The JetBench benchmark presented in this paper is an application benchmark written in C, for real-time jet engines thermodynamic calculations. It is a multithreaded application for shared memory architectures. The benchmark is based on OpenMP [2], and could be seamlessly ported to any platform supporting it. The

benchmark provides user the flexibility to specify custom workload, that could be a real flight profile with deadlines. The benchmark records the time consumed in calculating individual data points, and reports the miss of deadlines. The benchmark is scalable to theoretically any number of cores and could be used as a tool to measure an operating system's scheduling characteristics. This paper is divided into five sections. The following section overviews the related work in the area of embedded benchmarking, section 3 and 4 detail the proposed benchmark characteristics, and results based on a multicore architecture. Finally the last section forms the conclusion.

2 Related Work

With the current drive towards multicore platforms, standard APIs like OpenMP, POSIX [3] and Message Passing Interface (MPI) [4] have facilitated the development of multicore threaded applications. Multicore platforms have been widely applied in the real-time systems to achieve higher throughput and lower power consumption.
The embedded system community has long been using non-embedded benchmarks such as SPEC [5], Whetstone [6], Dhrystone [7] and NAS parallel benchmarks [8], to evaluate the performance of the target systems. A limited number of benchmarks are specifically designed for the embedded system evaluation.

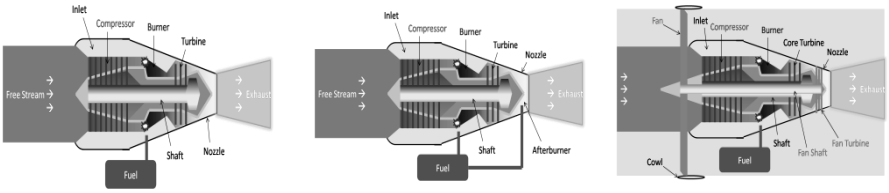
One of the few embedded system specific benchmark suites is the Embedded Microprocessor Benchmark Consortium (EEMBC) benchmark tools suite comprising of algorithms and applications targeting telecommunication, networking, automotive, and industrial products. A recent addition of a so called MultiBench [9] suite has realized the performance evaluation of shared memory symmetric multicore processors. These benchmarks could be targeted to any platform supporting POSIX thread library, and are delivered as customizable set of workloads, each comprising of one or more work items. Although computationally rich and extensive the benchmarks by no means provide real time performance statistics of the system, and for such applications EEMBC has two applications in a separate single core benchmark suite called AutoBench [10]. This benchmark suite comprises of real time applications such as 'Angle to Time Conversion' and 'Tooth to Spark' [11]. The Angle to Time Conversion application simulates an embedded automotive application, where the processor measures the real-time delay between pulses sensed from the gear on the crankshaft. Then it calculates the Top Dead Center (TDC) position on the crankshaft, computes the engine speed, and converts the tooth wheel pulses to crankshaft angle position. The Tooth-to-Spark application simulates an automotive application that processes air/fuel mixture and ignition timing in real-time. Another real-time single core embedded benchmark is PapaBench [12], that is based on a unmanned aerial vehicle (UAV) control software for AVR and ARM microcontroller systems. The benchmark provides the worst case execution time computation which is useful for systems scheduling analysis. Guthaus et al. [13] presented the MiBench embedded benchmark suite. This benchmark suite is a single core, non real-time implementation of 35 applications in the areas such as automotive/industrial, consumer, office, network, security, and telecommunication. As all of the above mentioned benchmarks either are not using threaded implementation or are not real-time applications, a more specific benchmark suite

addressing the two issues altogether is developed by Express Logic Inc., i.e. the so called 'Thread-Metric' benchmark suite [14]. The tool is specifically designed to measure a real-time operating system's (RTOS) capability to handle a threaded application. The benchmark is not a multiprocessor implementation as the thread model executes in a round-robin fashion and is useful to explore real-time context switching and memory management capabilities of an RTOS.

The related research in the embedded benchmarking area is pointing to the need of a more specific multicore real-time benchmark suite, capable to instrument performance characteristics of shared memory architectures. The following section introduces and overviews the JetBench benchmark application, an Open-Source tool for real-time, multiprocessor embedded architectures.

3 Benchmark Characteristics

The JetBench application is composed of thermodynamic calculations based on three types of jet engines, i.e. 1) TurboJet, 2) Turbojet with afterburner, and 3) a Turbofan engine (See Fig. 1). The application contains parameters specific to the said models as described in the NASA's EngineSim application [15]. The benchmark allows a user defined input flight profile to be simulated containing speed, altitude, throttle, and deadline time, while in response to that, the processing time for various thermodynamic calculations is monitored and reported (See Fig. 2).



(a) Turbojet (b) Jet with Afterburner (c) Turbofan

Fig. 1. Three different Jet Models used in JetBench (Adapted from [15])

An overview of the thermodynamic calculations used in the benchmark application is given in Appendix.

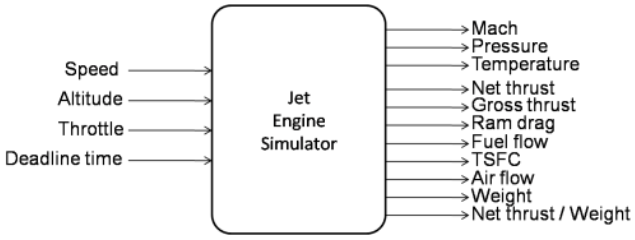


Fig. 2. JetBench Application I/O Parameters

In contrast to a synthetic benchmark, an application benchmark such as JetBench, is a realistic representation of the actual workload; however there are some deviations one has to apply to allow portability of the application on various platforms, which are discussed as follows. Generally, all real applications require a significant amount of I/O operations, which if were implemented in the benchmark would have restricted its portability [16]. Therefore the I/O performance of a platform can not be evaluated through the proposed benchmark. Secondly, as the application has to get executed in a target time period, excessive computations could have caused the benchmark to perform poorly on majority of low end systems. To avoid this problem the JetBench application covers a limited number of typical thermodynamic calculations used in jet engines. As a consequence of the restricted workload of the computations, it may seem small enough to high-end multicore systems that their actual performance may not be reported well, as in contrast to a low end multicore platform. A more detailed analysis of the benchmark on a number of cores is given in the following section.

The JetBench application not only provides the user with an overview of the real-time performance of the system, but could also be used to discover optimum number of threads to achieve desired performance. The JetBench benchmark is mainly comprised of ALU centric operations such as integer/double multiplication, addition, and division for the computation of exponents, square roots, and calculations such as value of pi and degree-to-radian conversion. All these operations are based on real thermodynamic equations and operations required for a jet engine control unit. The benchmark structure is composed of 88.6% of the parallel portion as reported by thread analysis tools, and is described in the pseudo code given in Fig. 3 and the threading diagram in Fig. 4.

```
JetBench Pseudo Code
Inputs: Engine Type
Data File Defining Speed, Altitude, Throttle, Deadline
Initialization:
      Set Default Parameters
      Select Engine Type
      Open data file
Parallel Section:
      Calculate Pi
      Read an input data point
      Calculate:
                Environment variables
                Thermodynamic parameters
                Engine geometry
                Engine performance
      Print Results
      If not EOF goto Parallel Section
      Print Results
      End
```

Fig. 3. Pseudo code of the application

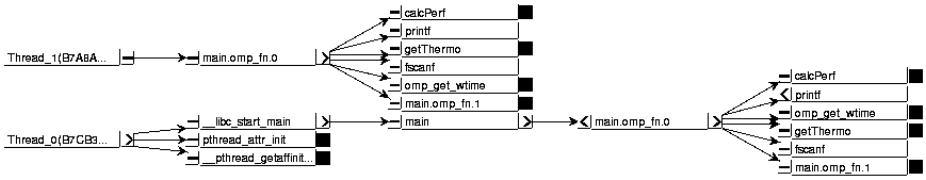


Fig. 4. JetBench Thread Structure

4 Results

To analyze the scalability of the benchmark, un-optimized executions of the application were carried out on shared memory multicore platform based on sixteen x86 CPUs running at 20MHz. The platform was simulated on Simics full system simulator [17], running Linux kernel 2.6.15 including symmetric multiprocessing support. The input dataset comprised of 30 data points and calculation deadlines were uniformly set as 9 sec. As the platform is running at a low clock frequency i.e. 20 MHz, a single thread per core was executed. The benchmark output timing per input data point is shown in Fig. 5. The graph shows normalized timing values against the set deadline time, i.e. 9 sec in this instance, which enables one to compare execution rate instead of execution time. It is worth noting that the execution rate is inconsistent for all the cores, also the rate decreases with the increase in number of cores. The reason behind is that the application is not prioritized statistically by the user but has been prioritized by the kernel itself. Secondly for any application increasing the number of threads beyond a certain level actually decreases performance since thread handling overhead will surpass the per thread execution time. This phenomenon is more observable, when running multiple threads per core where context switches depreciate the performance after a certain level of parallelism.

It can be observed from Fig. 6 and 7 that the overall execution time for the application is around 230 sec for a 4 core machine; however the 8 core machine offers a minimum number of missed deadlines, i.e. 2. This is due to the fact that although for 8 cores platform, threading overhead is higher than for the four core machine, which also effects the computation time per thread. But for 4 cores or less the CPU workload has exceeded the available resource and therefore resulted in missing more deadlines than the later. The output from the benchmark execution thus allows the user to analyze the impact of threading on a particular platform and could be helpful in the process to decide optimal number of cores as well as OS scheduling characterization.

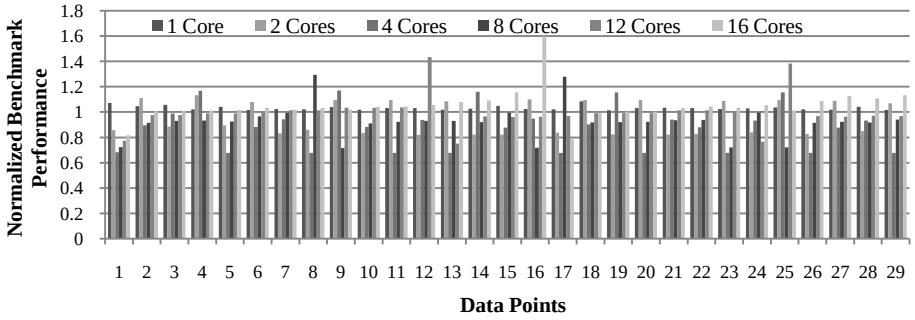


Fig. 5. Application execution rate for different number of cores

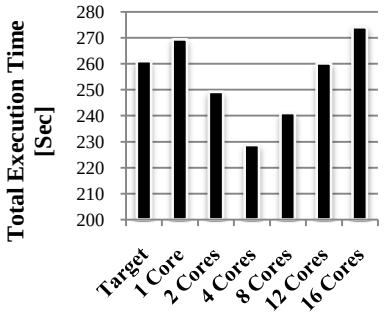


Fig. 6 Application execution time

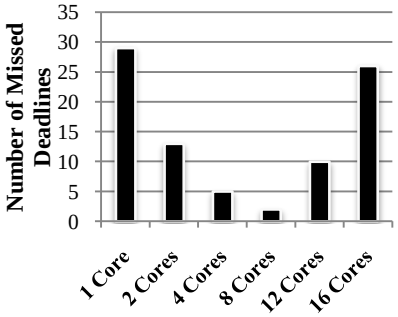


Fig. 7 Missed Deadlines

To validate the phenomenon of performance degradation with an increase in number of threads, a more detailed analysis of the benchmark based on an Intel Dual Core machine [18] was carried out (see Fig. 8). The benchmark is executed for up to 8 threads and processing speedup was calculated using Amdahl's law [19] and Gunther's law (or alternatively termed as Universal Scalability Law (USL))[20-22].

$$Speedup = \frac{1}{(1-p) + \frac{p}{N}}, \quad (1) \qquad \textbf{Amdahl's Law}$$

$$Speedup = \frac{N}{1 + s(N-1) + kN(N-1)}, \quad (2) \qquad \textbf{Gunther's Law}$$

where
p = Parallel fraction of the program
s = Serial fraction of the program
k = Delay associated with concurrency
N = Number of processors

Amdahl's law is useful in the situations to set an upper limit for the performance gain with increase of parallelization, this however does not take into account the drawbacks of aggressive parallelization such as excessive cache coherency delays, instruction execution, and thread scheduling delays etc. On the other hand Gunther's

law provides a more realistic picture in such situations. The results shown in Fig.8 complement the results in Fig. 6, as the throughput tends to decrease with the increase of parallelism beyond a certain limit, which however varies from platform to platform.

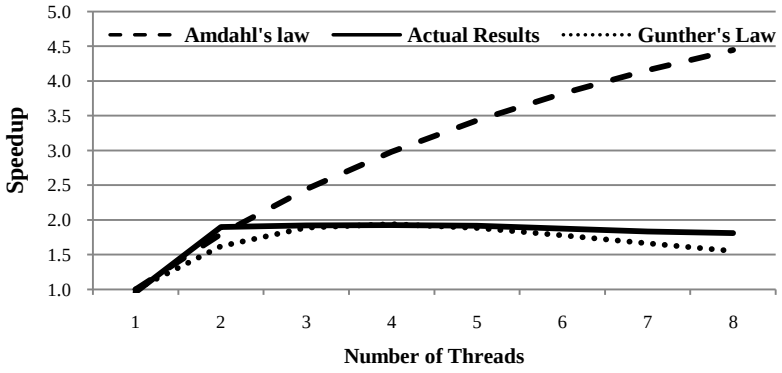


Fig. 8. Comparison of actual speed-up against Amdahl's law and Gunther's law

5 Conclusion

In this paper, Jetbench an open-source, real-time multicore application benchmark has been presented. The application is designed to be platform independent by avoiding target specific libraries and hardware counters and timers. The application comprises of thermodynamic calculations of a jet engine, and processes user defined input data points with custom deadlines. The benchmark application was tested on a 16 core platform and has demonstrated its usefulness for deciding optimal number of threads, and provided timing information that could be used to deduce an estimate of CPU core utilization and the operating system's real-time behaviour.

Future work will include the testing of the benchmark on various architectures with and without thread prioritization. Also the application's behaviour on an RTOS based platform is to be observed.

JetBench is available from http://jetbench.sourceforge.net/.

References

[1] G. Morton, "MSP430 Competitive Benchmarking," Texas Instruments 2005.

[2] L. Dagum, R. Menon, and S. G. Inc, "OpenMP: an industry standard API for shared-memory programming," *IEEE Computational Science & Engineering,* vol. 5, pp. 46-55, 1998.
[3] U. Drepper and I. Molnar, "The native POSIX thread library for Linux," *White Paper, Red Hat Inc.,* 2003.
[4] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Computing,* vol. 22, pp. 789-828, 1996.
[5] J. Uniejewski, "SPEC Benchmark Suite: Designed for today's advanced systems," *SPEC Newsletter,* 1989.
[6] R. P. Weicker, "An overview of common benchmarks," *Computer,* vol. 23, pp. 65-75, 1990.
[7] R. P. Weicker, "Dhrystone: a synthetic systems programming benchmark," *Communications of the ACM,* vol. 27, pp. 1013 - 1030 1984.
[8] H. Jin, M. Frumkin, and J. Yan, "The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance," NASA Ames Research Center, 1999.
[9] S. Gal-On and M. Levy, "Measuring Multicore Performance," *Computer,* vol. 41, pp. 99-102, 2008.
[10] P. Leteinturier and M. Levy, "The Challenges of Next Generation Automotive Benchmarks," *Journal of Passenger Car: Electronic and Electrical Systems,* vol. 116, pp. 155-160, 2007.
[11] L. A. Zadeh, *Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers by Lotfi A. Zadeh* vol. 6: World Scientific, 1996.
[12] F. Nemer, H. Cassé, P. Sainrat, J. P. Bahsoun, and M. De Michiel, "Papabench: a free realtime benchmark," in *6th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis*, Dresden, Germany, 2006.
[13] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *4th IEEE International Workshop on Workload Characterization (WWC 2001)*, Austin, Texas, 2001, pp. 184-193.
[14] "Measuring Real-Time Performance of an RTOS ": Express Logic Inc.
[15] "EngineSim Version 1.7a ": NASA Glenn Research Center.
[16] M. Berry, D. Chen, P. Koss, D. Kuck, S. Lo, Y. Pang, L. Pointer, R. Roloff, A. Sameh, and E. Clementi, "The Perfect Club benchmarks: Effective performance evaluation of supercomputers," *International Journal of High Performance Computing Applications,* vol. 3, pp. 5-40, 1989.
[17] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *IEEE Computer,* vol. 35, pp. 50-58, 2002.
[18] Intel, "Intel Concurrency Checker v2.1," Intel Corporation, 2008.
[19] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *AFIPS Joint Computer Conferences*, Atlantic City, New Jersey 1967, pp. 483-485.
[20] N. J. Gunther, "A Simple Capacity Model of Massively Parallel Transaction Systems," in *CMG Conference*, San Diego, California, 1993, pp. 1035-1035.
[21] N. J. Gunther, *Guerrilla Capacity Planning: a Tactical Approach to Planning for Highly Scalable Applications and Services*: Springer-Verlag New York Inc, 2007.
[22] N. J. Gunther, "Unification of Amdahl's law, LogP and other performance models for message-passing architectures," in *International Conference on Parallel and Distributed Computing Systems*, Phoenix, AZ, 2005, pp. 569-576.

Appendix: Thermodynamic equations

With reference to the Fig. 1, thermodynamic calculations [15] covered in the benchmark are given as follows.

Notations:

- Point 0 is the free stream conditions
- Point 1: the inlet entrance
- Point 2: compressor entrance
- Point 3: compressor exit
- Point 4: turbine entrance
- Point 5: turbine exit
- Point 6: nozzle throat

Inlet performance (0->2)	
$\frac{Tt_2}{Tt_0} = 1$	Inlet Temperature ratio
$\frac{pt_2}{pt_0} = \eta_i$	Inlet Pressure ratio for Mach < 1, where η_i is the inlet efficiency factor
$\frac{pt_2}{pt_0} = \eta_i(1 - 0.075[M-1]^{1.35})$	Inlet Pressure for Mach > 1, where η_i is the inlet efficiency factor
$D_{spill} = K(\dot{m}_i[V_1 - V_0] + A_1[p_1 - p_0])$	Spillage Drag for inlet, where **K** is the lip suction factor, $\dot{\mathbf{m}}_\mathbf{i}$ is the inlet mass flow rate, **V** is the velocity, **A** is the area, and **p** is denoting the pressure.
Compressor thermodynamics (2->3)	
$CPR = \frac{pt_3}{pt_2} \geq 1.0$	Compressor Pressure ratio
$\frac{Tt_3}{Tt_2} = \left(\frac{pt_3}{pt_2}\right)^{(\gamma-1)/\gamma}$	Compressor Temperature ratio, where γ is the ratio of specific heats

$CW = \frac{c_p Tt_2}{\eta_c}\left[CPR^{(\gamma-1)/\gamma} - 1\right]$	compressor work per mass of airflow, where $\boldsymbol{\eta_c}$ is the compressor efficiency factor and $\boldsymbol{c_p}$ the specific heat
Burner thermodynamics (3->4)	
$BPR = \frac{pt_4}{pt_3} = 1$	Burner Pressure Ratio
$\frac{Tt_4}{Tt_3} = \frac{1 + \frac{f.\eta_b.Q}{cp.Tt_3}}{1+f}$	Burner Temperature Ratio, where , $\boldsymbol{f}$ is the fuel to air mass flow ratio, $\boldsymbol{Q}$ is the heat release, $\boldsymbol{\eta_b}$ is the burner efficiency factor.
Turbine thermodynamics (4->5)	
$TPR = \frac{pt_5}{pt_4} = \frac{Tt_5 5^{(\gamma-1)/\gamma}}{Tt_4}$	Turbine Pressure Ratio
$TW = \eta_t c_p Tt_4\left(1 - TPR^{(\gamma-1)/\gamma}\right)$	Turbine Work Per Mass Of Airflow, where $\boldsymbol{\eta t}$ is the turbine efficiency and $\boldsymbol{c_p}$ is the specific heat.
Nozzle thermodynamics (5->6)	
$NPR = \frac{pt_8}{pt_5} = \left(\frac{Tt_8}{Tt_5}\right)^{\gamma/(\gamma-1)} = 1$	Nozzle Pressure and temperature ratios
$V_e = V_8 = \sqrt{2c_p Tt_8 \eta_n [1 - \{1/NPR\}^{(\gamma-1)/\gamma}]}$	Exit velocity, where $\boldsymbol{\eta_n}$ is the nozzle efficiency
Output calculations	
$M = \frac{V_0}{a_0} = \frac{V_0}{\sqrt{\gamma R T_0}}$	Mach Number, where $\boldsymbol{V_0}$ is the aircraft speed, $\boldsymbol{a_0}$ is the speed of sound and $\boldsymbol{R}$ is the gas constant
$ts_0 = 518.6 - 3.56\,\frac{altitude}{1000}$, $ps_0 = 2116.\frac{ts_0}{518.6}^{5.256}$.	Stratospheric Temperature and pressure for altitude < 36152 feet

$ts_0 = 389.98$, $ps_0 = 473.1376 * e^{(36000-altitude)/20805.433}$	Stratospheric Temperature and pressure for 36152< altitude < 82345 feet
$ts_0 = 391.625 . \frac{altitude-82345}{1000}$, $ps0 = 51.96896 * \left(\frac{ts_0}{389.98}\right)^{-11.388}$.	Stratospheric Temperature and pressure for altitude > 82345 feet
$\dot{m}_f = f\dot{m}_a$ where, $f = \frac{\left(\frac{Tt_4}{Tt_3}\right)-1}{(\eta_b Q/c_p Tt_3)-(Tt_4/Tt_3)}$.	Fuel mass flow rate, where $\dot{m}_a$ is the airflow rate, η_b is the burner efficiency, f is the fuel to air ratio and Q is the fuel heating value.
$EPR = \frac{pt_8}{pt_2} = \frac{pt_3}{pt_2}.\frac{pt_4}{pt_3}.\frac{pt_5}{pt_4}.\frac{pt_8}{pt_5}$, $ETR = \frac{Tt_8}{Tt_2} = \frac{Tt_3}{Tt_2}.\frac{Tt_4}{Tt_3}.\frac{Tt_5}{Tt_4}.\frac{Tt_8}{Tt_5}$, $Tt_8 = Tt_2.ETR$, $pt_8 = pt_2.EPR$, $Tt_0 = Tt_2 = T_0(1 + 0.5[\gamma - 1]\frac{{V_0}^2}{{a_0}^2})$, $pt_0 = p_0 \frac{Tt_0^{\gamma/(\gamma-1)}}{T_0}$.	Thrust Specific Calculations where, ***EPR*** is the engine pressure ratio, and ***ETR*** is the engine temperature ratio.