



Dear reader of this benchmark

The following general comments should be noted on C Code benchmarking:

- Our goal with this presentation is to support our claim that we have a very High Level Language (in this case 'C') suitable architecture.
- The results may vary from application to application, but in average you will find AVR among the most code efficient architectures in the market.
- There does not exist any standard to measure C Code efficiency, hence we have collected real applications from customers. There is, however, an emerging standard for measuring C Code efficiency. The EEMBC (EDN Embedded Microprocessor Benchmark Consortium) are currently developing a benchmark suite of real and synthetic applications written in C which we will look into when it becomes available.
- Since not all code could be compiled for all compilers, all the indexes are relative to AVR. Only the applications that could be compiled for both two applications are summed, and the ratio between the code sizes is displayed.
- Since not all code could be compiled for all compilers, all the indexes are relative to AVR. Only the applications that could be compiled for both two applications are used, and the ratio between every two applications is computed. The averaged ratio is displayed in the figure.
- Most of the compilers used are from IAR. The advantage of using the same compiler company for most of the processors is that the difference in the code size does not depend on the global optimization which all compilers will benefit from, but to a larger degree of architectural differences.
- Some of the compilers exist in later versions, so the results might be different for some of the microcontrollers.

The AVR C Compiler is, compared to many of our competitors, a relatively young compiler. The AVR C Compiler still gains significant code size decrease in every new release of the compiler.

By writing the code with the AVR architecture a smaller code size can be achieved. The code in the benchmark is NOT optimized for AVR, but the result is still very good !

Enjoy your reading



AVR[®] ENHANCED RISC

AVR[®]

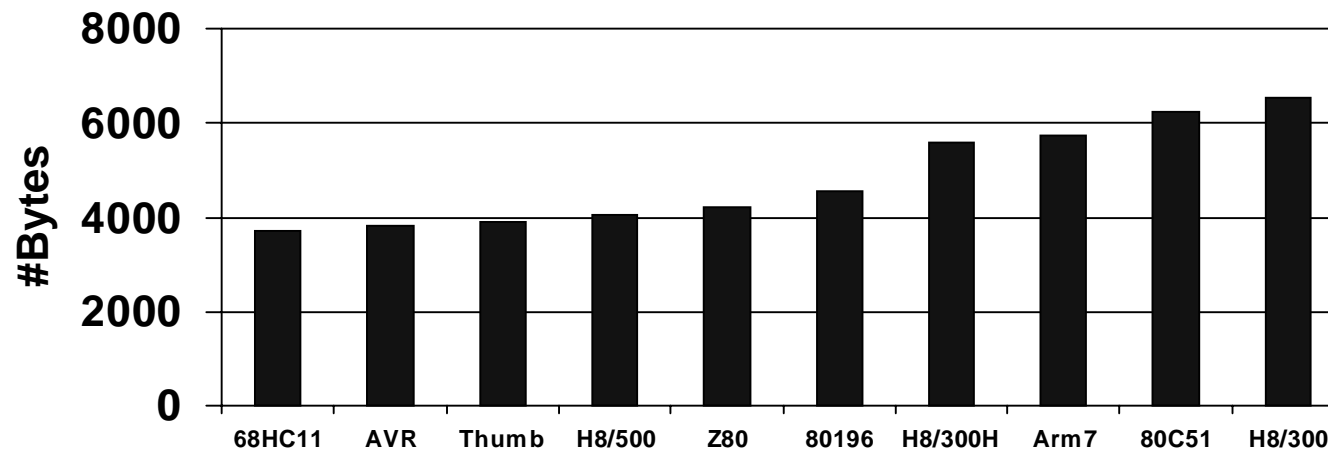
C Code Benchmarks

C Code Benchmarks

- **Nine applications**
- **Based on customer code**
- **Various application areas**
- **Byte usage in individual applications**
- **Summarized results reported as**
 - **Normalized Accumulated Results**
 - **Averaged Normalized Results**

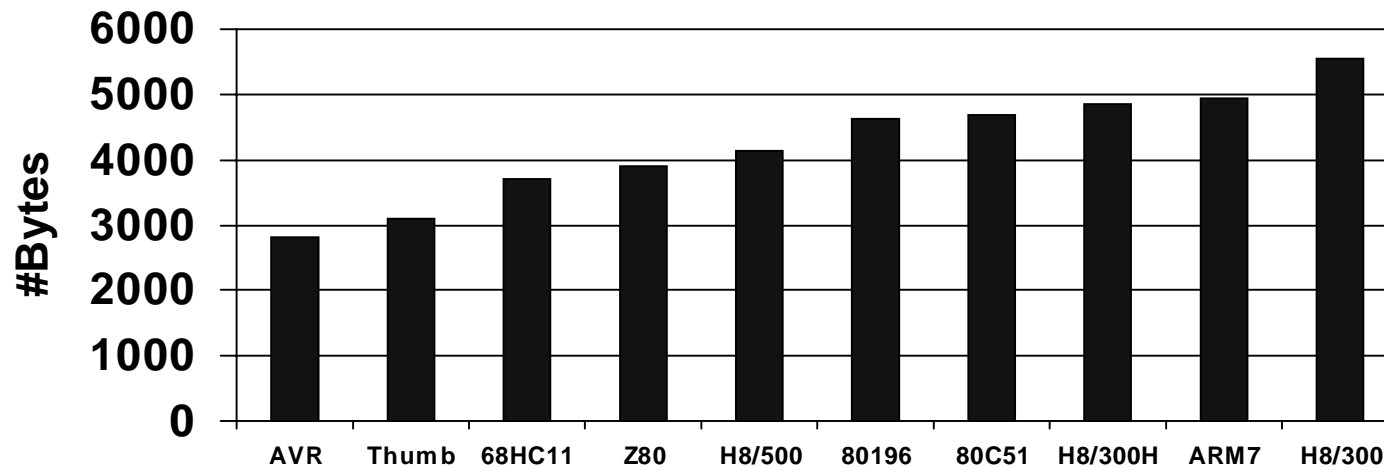
Pager protocol

- Three layer protocol
- Includes simple driver



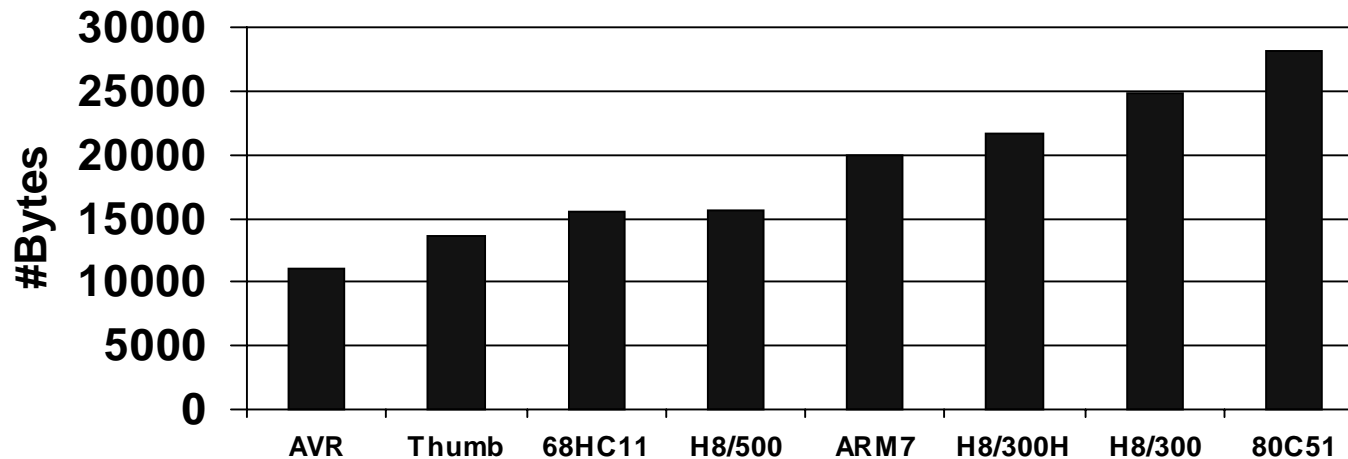
Analog Telephone I

- SIM Interface
- Parts of display driver



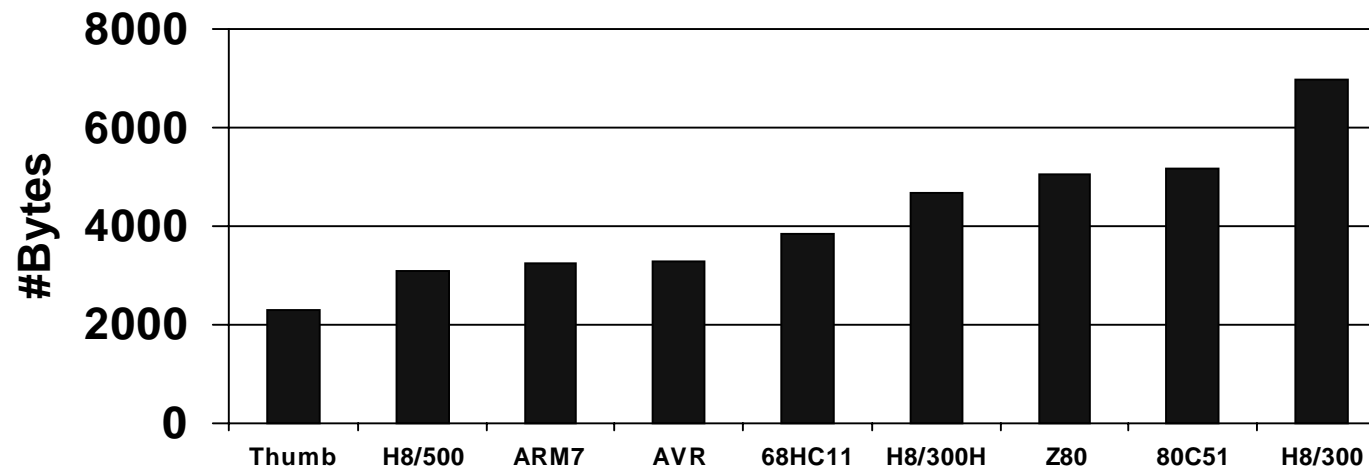
Analog Telephone II

- Automatically generated code
- State Machine based



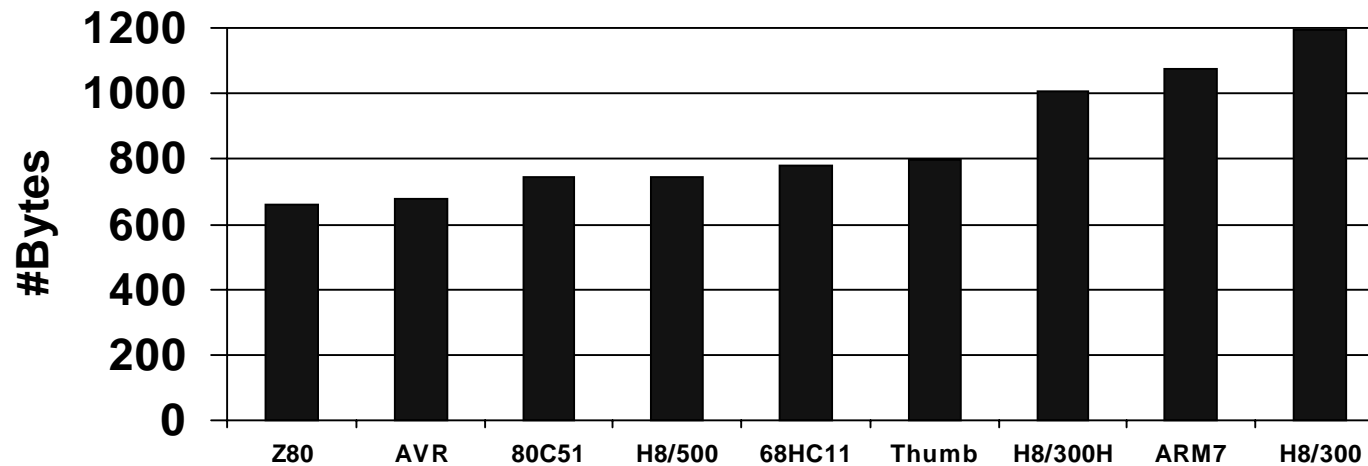
Reed-Solomon

- Reed-Solomon Encoder/Decoder



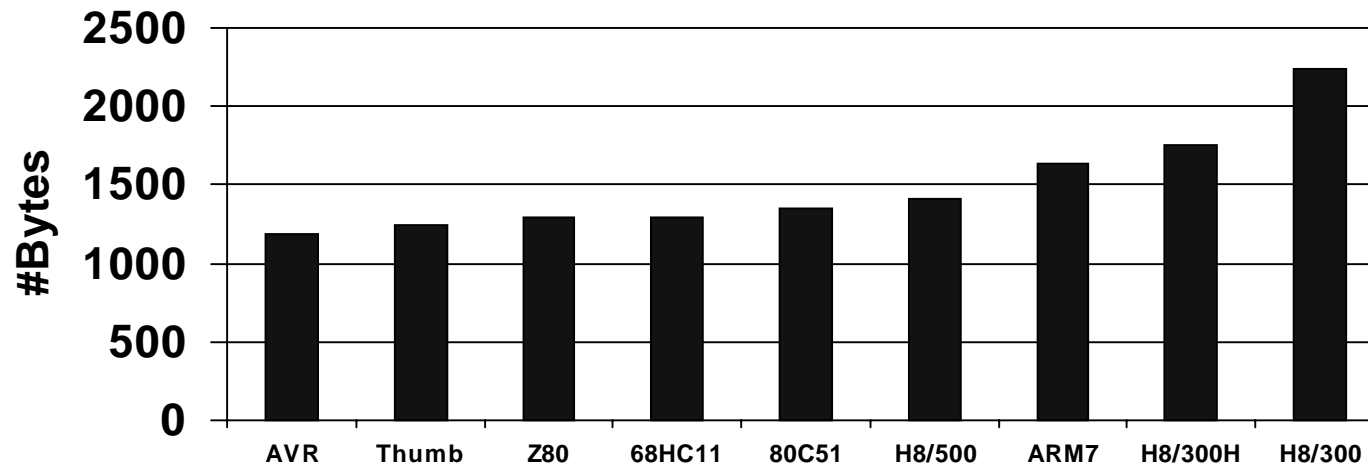
Car Radio Control

- Skeleton application
- Control Flow and Bitfields



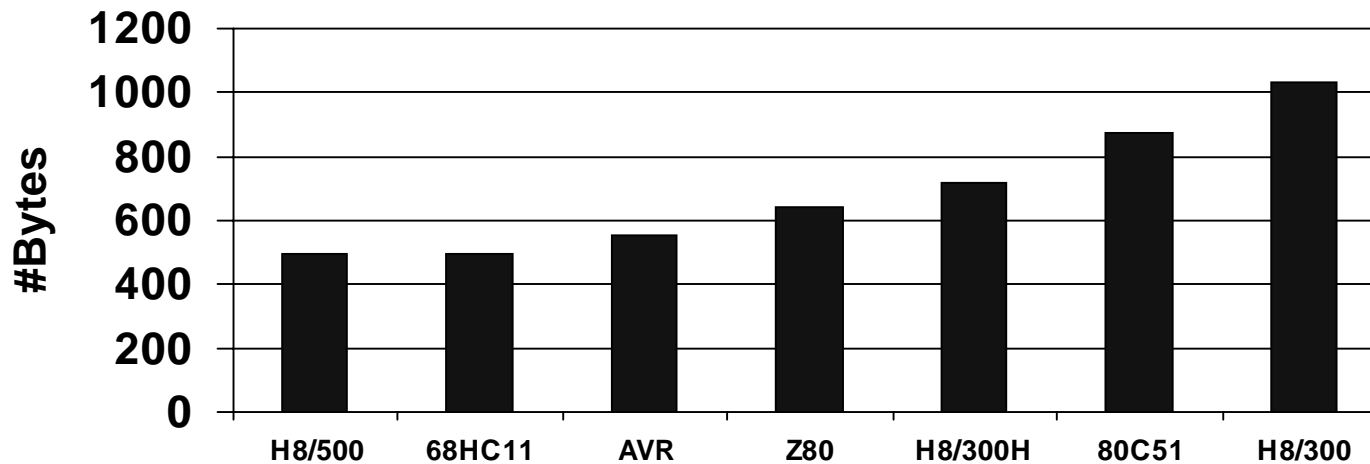
C Bitfields

- Benchmark code from customer
- 8 and 16 bit bitfield variables



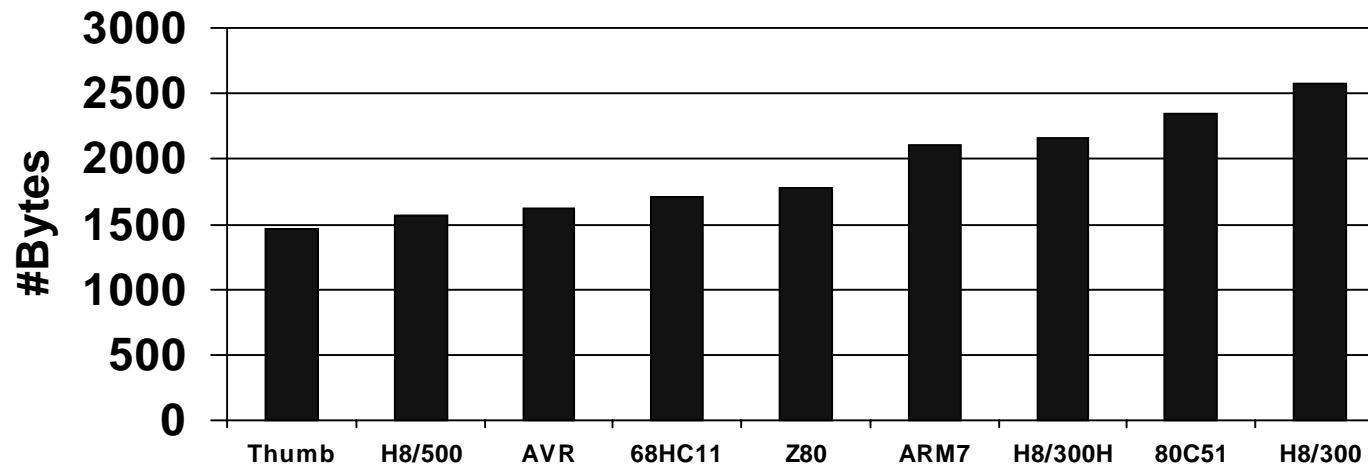
Analog Telephone III

- Representative collection of routines from an analog telephone application



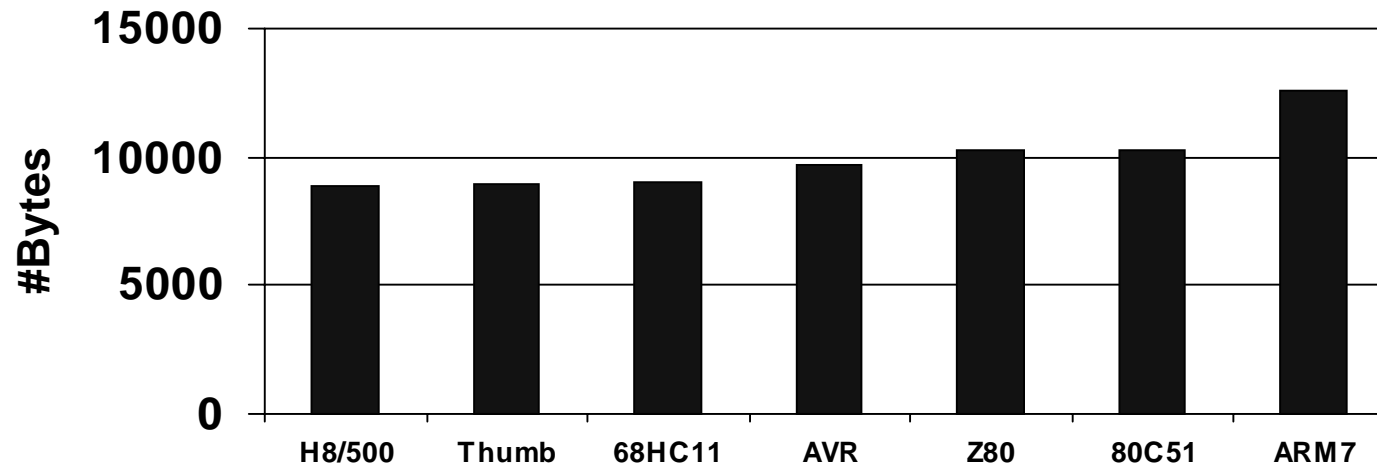
DES Algorithm

- Encryption/Decryption algorithm



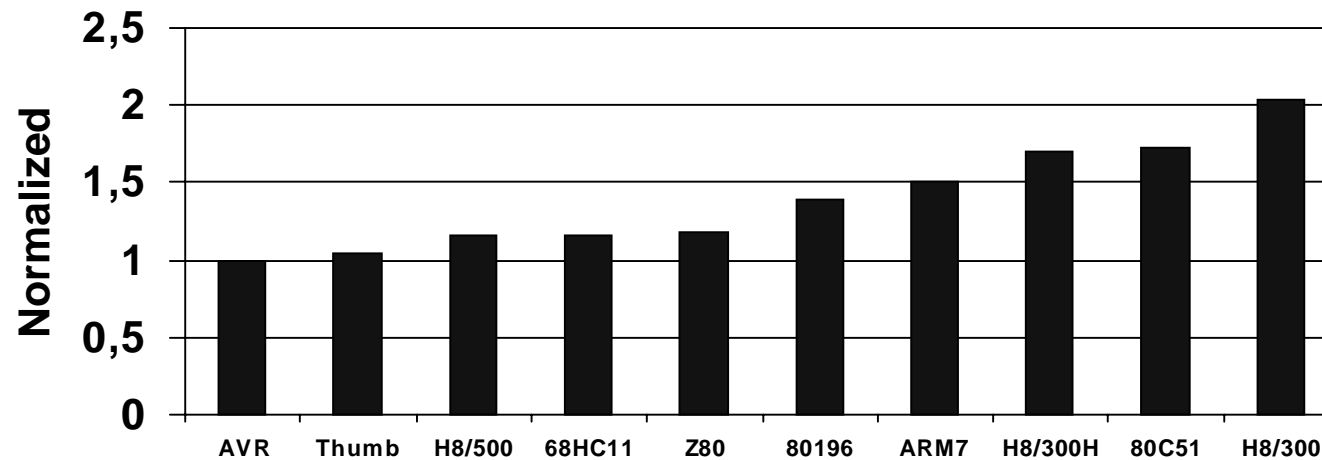
Navigation Application

- Complete application
- Communication, measurements, computations



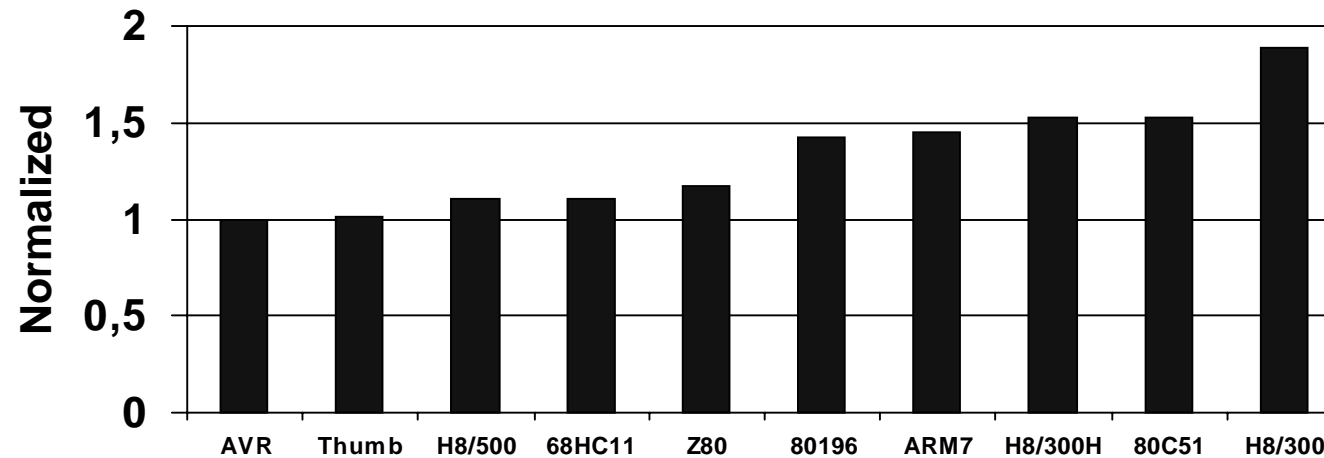
Accumulated over all Benchmarks

- Indexes based on accumulated sizes
- Large applications counts more than small



Normalized over all Benchmarks

- Averaged indexes from all applications
- All applications counts evenly



Summary

- **Nine C Code Benchmarks from various application areas**
- **No single microcontroller best for all applications**
- **AVR in the top range for all the applications**

Compilers Used

- **AVR: IAR ICCA90 version 1.40**
- **80C51: IAR ICC8051 version 5.20**
- **Thumb: ARM tcc version 1.02b**
- **ARM: ARM armcc version 4.66b**
- **80196: IAR icc196 version 5.20a**
- **Z80: IAR iccz80 version 4.03a**
- **H8/300(H): IAR icch83 version 3.22**
- **H8/500: IAR icch8500 version 2.92g**
- **68HC11: IAR icc6811 version 4.20B**