User-Friendly Methodology for Automatic Exploration of Compiler Options: A Case Study on the Intel XScale Microarchitecture

Haiping Wu Eunjung Park Long Chen Juan del Cuvillo Guang R. Gao

University of Delaware
Department of Electrical and Computer Engineering
Newark, Delaware 19716, U.S.A
{hwu, epark, lochen, jcuvillo, ggao}@capsl.udel.edu

Abstract

Finding an optimized combination of compiler options that benefits the most a given embedded application is a challenge for most application developers. It is only with a deep understanding of the application at hand and a fairly good knowledge of the compiler features that a programmer can achieve the desired results in terms of performance, power consumption and code size out of an application.

We have developed a practical methodology for automatic exploring compiler options (UMECO) to solve the problem mentioned above. This paper reports a case study of this methodology on the Intel XScale microarchitecture. Based on practical experimentation, we enhance KCC, our research compiler infrastructure, with an extended user interface that the users can provide advice to. All are controlled by a reduced set of compiler flags. We also demonstrate a compiler trade-off strategy based on experimental results for a set of well known embedded benchmarks.

Keywords: Compiler option, Performance, Power, Codesize, Microarchitecture

1. Introduction

The specific features and requirements of embedded applications bring about a new challenge to the traditional compiler design methodology. These challenges have resulted in numerous studies that focus on improving compiler technology to meet the specific requirements of embedded systems [3, 4, 7, 9].

Nowadays, modern compilers for embedded systems support irregular microarchitectures (i.e., digital signal processors, network processors, micro-controller units), complex instruction sets (i.e., application specific instruction sets) and capture architecture specific optimization features (i.e., parallelism in multi-core or multi-function units).

To handle such a broad spectrum of possibilities, compilers supply a large number of optimization options. Unfortunately, it is the application developers' responsibility to find a suitable combination of options for each specific application. Meanwhile, finding a combination of compiler options such that the compiled program meets the specifications is not a trivial task. The problem becomes even more complicated when a trade-off between performance (execution time), power consumption and code size (bytes of the text section) is added to the list of requirements. From the application developers' perspective, it would be desirable to have a simple compiler-user interface that based on an application profile could come up with an optimal combination of compiler options, allowing developers to concentrate on other aspects of the development.

Therefore, there is a strong requirement moving the study toward the methodologies that find an optimal combination of compiler options for different applications running on a specific architecture.

We have developed a practical methodology for automatic exploring the compiler options (*UMECO*) [6] which can be applied directly to embedded applications. The strategy behind the methodology is to first find an optimized combination of compiler options by experimental measurement for a set of typical applications. Armed with this knowledge, the compiler has then the ability to automatically make a good trade-off between performance, power consumption and code size for applications in the same domain.

This paper is a progress report on part of the *UMECO* work on the Intel XScale microarchitecture, for which we setup a hardware testbed using the Intel XScale 80200 Evaluation Board (80200EVB). Our experimental platform allows us to measure the power consumption and the execution time for a DSP kernel test suite. We use our KCC compiler infrastructure [1] and a special developed run-time library to adjust and monitor execution factors such as the core frequency and voltage.

Our study comprises of four steps. First, we measure the execution time, power consumption and code size for a DSP kernel suite compiled with KCC using the $-O3$ option. This information is regarded as the baseline of our analysis. Second, we conduct additional measurements using single compiler options this time. A subsequent analysis of the results drives the selection of a subset of options, which is then followed by additional measurements with combinations of the options just selected. Finally, we analyze the measurement results and create a *Candidate Set* in which each element is an optimized combination of selected options. We propose an application-driven compiler strategy that based on the *Candidate Set* automatically performs a reasonable trade-off between the execution time, power consumption and code size. The corresponding compiler-user interface is also developed. Using this interface, application developers can make a full use of the compiler optimization features without any knowledge of the specific compiler optimization options. Thanks to the interface and the set of optimized combinations that are found by means of experimental measurements, the underlying compiler shall use an optimized combination of options to compile the code thereafter.

The contributions of this paper are summarized as follows:

- We confirm the effectiveness of the *UMECO* methodology on a practical architecture.
- We develop a full system measurement platform for the Intel XScale microarchitecture.
- We develop an application-driven complier trade-off strategy.

The remainder of this paper is organized as follows. In Section 2, we outline the full system measurement platform. In Section 3 we present our methods of measurement and analysis of the power consumption, execution time and code size. The method to find an optimized combination of compiler options is also discussed in this section. The application-driven compiler trade-off strategy is described in Section 4. We present related work in Section 5. Some conclusions are given in section 6.

2. Experimental Platform

In this section, we describe the experimental platform used in our study. It is used to measure the execution time, power consumption and code size. The platform consists of hardware and software parts. The hardware side is shown in Figure 1. It includes the Intel XScale 80200EVB with an Intel XScale processor and measurement equipment. Figure 2 shows the software side, including the Wasabi Software Development Toolkit (SDT) [2], the KCC compiler, a run-time software library and a DSP kernel test suite.

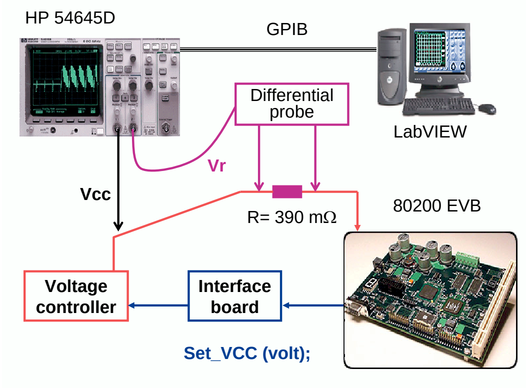

Figure 1. Hardware Testbed

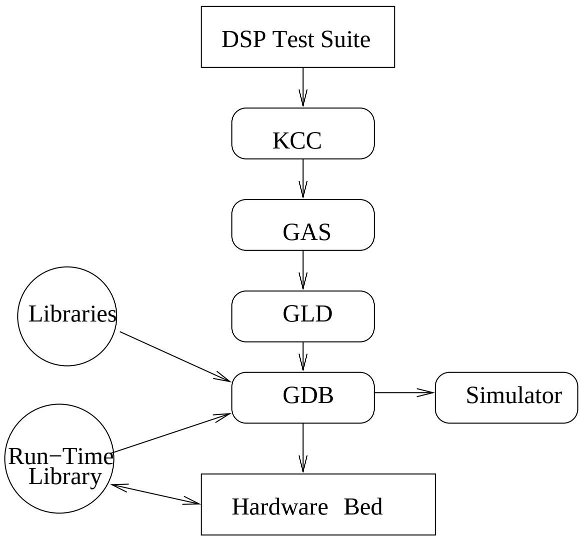

Figure 2. Toolchain Hierarchy

2.1. Intel XScale 80200EVB

The Intel XScale microarchitecture features a unique design optimized for low power consumption and high performance processing for a wide range of applications including DSP applications, Internet applications, handheld devices, networking and wireless equipments, and remote access servers.

The Intel XScale 80200 is the first processor based on the Intel XScale microarchitecture. It integrates an external bus and interrupt controllers around an ARM-compliant processor core. The 80200EVB used in the platform comprises of an Intel 80200 XScale processor, 32MB on-board SDRAM, 4MB flash memory and a variety of peripherals, see Figure 3.

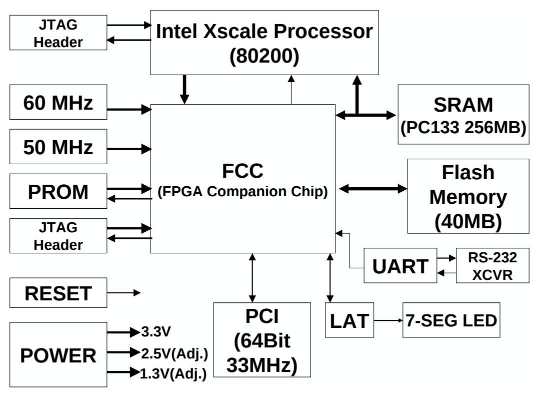

Figure 3. **80200EVB Architecture**

On the PCB board, removable power supply tracks are provided to separate the 80200's core power supply from the rest of the 80200EVB and insert an instrument resistor. For our measurements, we insert a 0.390Ω resistor and measure the voltage drop with a differential probe and an HP 54645D oscilloscope. A remote PC running LabView-based virtual instrument software is responsible for setting up the oscilloscope, acquiring the data and its final post-processing.

The XScale processor frequency can be adjusted from 333MHz to 733MHz. In addition, the 80200EVB is attached to a voltage controller that can regulate the core voltage from 1.5V to 0.7V. Both parameters are set by writing into a CPU special-purpose control registers.

2.2. KCC Compiler and Toolchain

The KCC compiler is used in experiments. This C compiler is based on the Open64 compiler suite, and has been re-targeted to the Intel XScale architecture. It is a cross compiler that runs on a Linux platform and generates code for the XScale platform. The KCC compiler features all the high level optimizations available in the Open64 compiler. The back-end code generator has been re-targeted to the XScale instruction set architecture. The object code produced by the KCC compiler is linked with other objects by GNU LD linker (provided by the Wasabi SDT). The executable code is executed on the actual evaluation hardware. At compile time, the user must provide a board-specific configuration file. For Intel 80200 board, the specification file is *lrh.specs*.

2.3. Run-Time Library

To measure the execution time and power consumption on the Intel XScale 80200EVB, we developed a run-time library written in ARM assembly language. Currently the library includes 5 modules:

- vcc0.s: Program the voltage supply controller by writing a value into the memory-mapped I/O port located at address 0x600000.
- cclkcfg.s: Program the frequency multiplier by setting a value into the core clock configuration register (CCLKCFG).
- pmnc.s: Reset the Performance Monitor Control register (PMNC) and enable some performance counters.
- led0.s: Program the LED Controller by writing a value into the memory-mapped I/O port located at address 0x500000. It is used to synchronize program execution with data acquisition.
- ccnt.s: Read the Clock Counter register (CCNT). CCNT is set to count every 64th processor clock cycle.

3. Measurement and Analysis

This section describes the method we follow to find optimized combinations of compiler options, using the experimental platform presented in Section 2.

3.1. Benchmark and Measurement Strategy

Currently, a DSP test suite with 8 kernels is used for this study. These programs are widely used in real DSP applications and other application domains and, therefore, are regarded as good representatives. Moreover, we use 4 kernels for the initial measurements and leave the other 4 for the purpose of result confirmation, see Table 1.

The execution time is measured in cycles using a function provided by our run-time library. The number of cycles C, is derived from the expression:

$$C = 64 \times ((CCNT_{end} - CCNT_{start})/Frequency)/10^2$$

It is a little more complex to measure the power consumption. As shown in Figure 1, the voltage supply and

DSP Kernel	Description	Usage
mm	matrix multiply	Measurement
vec_mpy1	vector multiply	Measurement
mac	dot product	Measurement
latsynth	Lattice Synthesis	Measurement
fir	FIR filter	Confirmation
iir1	IIR filter	Confirmation
codebook	Vocoder codebook search	Confirmation
jpegdct	JPEG discrete cosine transform	Confirmation

Table 1. **DSP Test Suite**

the voltage drop at the resistor are both captured by the oscilloscope. The acquired data are then processed on a remote computer. From the formula $I = V/R$, the current drawn through the resistor can be easily derived. Since the processor is in series with the instrumental resistance, the current drawn at the processor core is the same as that drawn through the resistor. Hence, we can use the expression $P = V \times I$ to calculate the power consumption of the core. Unlike power consumption and performance, it is trivial to obtain the code size directly from the binary file.

Once our platform is ready, we run the DSP kernels with different frequency and voltage setting as well as compiler optimization levels to observe the effects on the measured the execution time, power consumption, and code size. Then we look at the relationship between execution time and/or power consumption for a set of frequency and voltage values and specific compiler option. We notice that the relationships between execution time and/or power measured for each program holds regardless of the optimization level.

Based on this observation, we decided not to perform measurements using any other combinations of frequency and voltage, except for the default frequency (733MHz) and voltage supply (1.5V), which is used in the subsequent sections.

3.2. Baseline Measurement

For each measurement case, every combination of optimization options produces a triplet with the execution time in clock cycles, the power consumption in wattage, and code size in bytes. By adding the results for the 4 kernel programs, we obtain what we call the measurement suite performance, power and size factors as the sum of clock cycles, wattages and bytes, respectively. The triplet obtained for the measurement suite when the compiler option is *-O3* is regarded as our baseline, see Table 2.

O3 is the highest optimization level available in the KCC compiler. It turns on almost all of the KCC optimizations and usually improves the performance at the expense of compilation time. Most application developers are used to use *-O3* optimization option when compiling programs for both late-cycle testing and production use.

Kernel	Cycles	Average Power[W]	Code Size[B]
mm	45594642	0.461	1152
vec_mpy1	123812658	0.544	1572
mac	96093906	0.549	1564
latsynth	328695472	0.535	1476
Baseline = $<$594196678, 2.089, 5764 $>$			

Table 2. **Baseline Measurement Results**

Option	Description
o_1	INLINE:none
o_2	SWP:=ON -OPT:unroll_times_max=0
o_3	LNO:pure
o_4	SWP:=OFF
o_5	CG:hb_formation=OFF

Table 3. **The Selected Individual Options**

The motivation behind the definition of *-O3* triplet as the baseline is that we can find combinations of optimization options for which at least one factor in their corresponding triplets is smaller than the corresponding baseline factor.

3.3. Individual Option Selection

Because our motivation is to confirm the effectiveness of the *UMECO* methodology [6] in this case study, we do not adopt the time consuming formal method of option selection presented in the *UMECO*. Instead, we use the following criterion to randomly select individual options in our study:

When an option is selected, a factor in its resulting triplet must be smaller than the corresponding factor in all the remaining triplets.

We choose 5 options that satisfied the above condition among 30 optimization options in KCC, see Table 3. In particular, these options override *-O3* because they are turned off when *-O3* is used. The measurement results using these 5 individual options are shown in Table 4.

3.4. The Candidate Set

The next step in our analysis is to find candidate combinations for our trade-off strategy. A candidate combination is one for which a triplet factor is smaller than the corresponding baseline factor.

Option	Performance factor	Power factor	Size factor
o_1	594196419	2.078	5764
o_2	706590649	2.077	3720
o_3	594196394	2.094	5764
o_4	594196354	2.081	5764
o_5	594196461	2.083	5764

Table 4. **Measurement Results with Single Options**

In the process of finding candidate combinations, we create a *Candidate Set* using the algorithm outlined in Figure 4. Each element in the *Candidate Set* consists of a candidate combination and the corresponding triplet. The candidate combination of options is represented by a vector, $B = \langle o_1, o_2, ..., o_n \rangle$, where $o_i (i = 1, ..., n)$ corresponds to the ith option in Table 3. The value of o_i is either 1 or 0, with 1 meaning the ith option has been selected as a part of a candidate combination and 0 otherwise. To simplify the notation, we represent the triplet corresponding to a combination *B* as *B.triplet*. Note that the initial element in the *Candidate Set* is the baseline, that is, *O3* and *O3.triplet*. Table 5 shows the *Candidate Set* obtained from the DSP kernel suite.

```
for i = 1; i <= n
  do
    for each combination B of i options
      do
        measurement and form the B.triplet;
        if (the size factor in B.triplet > the size factor in the O3.triplet &&
            the performance factor in B.triplet > the performance factor in the O3.triplet &&
            the power factor in B.triplet > the power factor in the O3.triplet)
           break;
        send B and B.triplet to Candidate Set
     enddo
   enddo
```

Figure 4. **Algorithm of Generating the *Candidate Set***

4. A Compiler Trade-Off Strategy

Modern compilers have potential abilities to generate optimal performance codes for several different application domains. They use different combinations of optimizations to organize user codes for different applications. On the other hand, it is impossible for the compiler to know which domain an application belongs to, and thus what the optimal optimization options would be. This is the reason why compilers supply dozens of options for users and it is users' responsibility to choose the appropriate optimization options for their applications.

In this section, we propose a compiler trade-off strategy which is based on the *Candidate Set* generated in Section 3.4. We first present a new compiler-user interface based on our trade-off strategy and the describe the principles of the trade-off strategy.

Combination of options	Triplet
B1=$< 0, 0, 1, 0, 1 >$	<594196312, 2.082, 5764>
B2=$< 0, 0, 0, 1, 0 >$	<594196354, 2.081, 5764>
B3=$< 0, 0, 1, 0, 0 >$	<594196394, 2.094, 5764>
B4=$< 1, 0, 0, 0, 1 >$	<594196395, 2.084, 5764>
B5=$< 1, 0, 0, 0, 0 >$	<594196419, 2.078, 5764>
B6=$< 0, 0, 0, 0, 1 >$	<594196461, 2.083, 5764>
B7=$< 1, 0, 1, 1, 1 >$	<594196501, 2.080, 5764>
B8=$< 0, 0, 0, 1, 1 >$	<594196518, 2.085, 5764>
B9=$< 1, 0, 1, 1, 0 >$	<594196521, 2.087, 5764>
B10=$< 0, 0, 1, 1, 0 >$	<594196541, 2.085, 5764>
B11=$< 1, 0, 1, 0, 0 >$	<594196574, 2.096, 5764>
B12=$< 0, 0, 1, 1, 1 >$	<594196585, 2.082, 5156>
B13=$< 1, 0, 1, 0, 1 >$	<594196615, 2.080, 5764>
B14=$< 1, 0, 0, 1, 0 >$	<594196626, 2.080, 5764>
B15=$< 1, 0, 0, 1, 1 >$	<594196659, 2.082, 5764>
B16=*baseline*	<594196678, 2.089, 5764>
B17=$< 0, 1, 0, 0, 0 >$	<706590649, 2.077, 3720>
B18=$< 1, 1, 1, 0, 0 >$	<706590673, 2.059, 3720>
B19=$< 0, 1, 1, 0, 0 >$	<706590829, 2.098, 3720>
B20=$< 0, 1, 1, 0, 1 >$	<706590890, 2.090, 3720>
B21=$< 1, 1, 1, 0, 1 >$	<706590923, 2.085, 3720>
B22=$< 0, 1, 0, 0, 1 >$	<706590926, 2.089, 3720>
B23=$< 1, 1, 0, 0, 1 >$	<706591005, 2.101, 3720>
B24=$< 1, 1, 0, 0, 0 >$	<706591078, 2.077, 3720 >

Table 5. **Candidate Set Generated by DSP Suite**

4.1. A New Compiler-User Interface

Different from the traditional compiler interface, our interface has only a few simple options with the format as follows:

Compiler_name [*option_class_1*] [*option_class_2*]

There are two groups of option list in the new compiler-user interface: *option_class_1* is an option list that the application developer users to describe application-specific requirements to the compiler. *option_class_2* is the original option list that the compiler supplies. During the compilation, the first option group has priority over the second.

The formats and meanings of the options in *option_class_1* are described as follows:

- ***–p***=<$\boldsymbol{p}_1$,$\boldsymbol{p}_2$,$\boldsymbol{p}_3$> A priority list of performance(*P*), power(*W*) and code size(*S*), with p_i(i=1,2,3) being one of the three characters *P*, *W* and *S*. The list represents the priority order for the compiler to trade-off between performance, power and code size. For example, *–p*=<W, S, P> means the user cares most about minimizing the power consumption, followed by code size minimization, and least about improving the performance.

- ***–w=<c*$_1$*,c*$_2$*,c*$_3$*>*** Weighted coefficients corresponding to the performance, power and code size. c$_i$(i=1,2,3) is a fraction in [0,1] and their sum must $\leq$ 1. It represents the parameters to a balance algorithm in the compiler when it makes the trade-off among the performance, power and code size. The *–w* option overrides the *–p* option.

- ***–a=class*** An option to describe what category the application belongs to, such as network application (*class* is *N*), DSP application (*class* is *D*), wireless communication (*class* is *W*), etc.

The *option_class_2* is provided for compatibility purposes, since some of the standard compiler interface options are still needed, such as the options to describe the application name, binary name, and so on.

4.2. Trade-Off Strategy Based on Candidate Set

The principle of our trade-off strategy is described as follows. The compiler integrates a set of databases classified by application domains. Each application class has its corresponding *Candidate Set* obtained from experimental results. Table 5 shows the *Candidate Set* for DSP applications.

Three databases are derived from the *Candidate Set*: the performance database (having the minimum number of cycles), the power database (having the minimum number of wattages) and the code size database (having the minimum number of bytes). The elements in each database are sorted by the key value.

If the user's concern is only one metric, either performance, power consumption, or code size, the compiler simply finds the database, according to which application domain the user program belongs to, and selects the first element from the database.

On the other hand, if the user has to make a trade-off between all the three metrics and supplies weighted coefficients, the compiler will normalize the values of all triplets first, and then trade-off based on the normalized results.

The reason of the normalization is that the values of the three elements in a *triple* differ greatly. For example, the wattage values are much smaller than the values representing clock cycles (see the Table 2, Table 4 and Table 5). Without the normalization, the effect of the power consumption may be overwhelmed by other metrics, even if a big weighted coefficient is given to the power factor.

Currently, we adopt a simple normalization mechanism in our strategy. We use the coefficients 10^{-5}, 2*10^3 and 1 as the normalization factors to multiply the values of the number of cycles, the wattages and the bytes, respectively.

The compiler obtains the options from the compiler-user interface or uses default options if no application-specific option is provided. It first uses the ***–a=class*** option to find the set of corresponding databases. Then, in the selected database, the trade-off is made based on the ***–p=<p*$_1$*,p*$_2$*,p*$_3$*>*** and the ***–w=<c*$_1$*,c*$_2$*,c*$_3$*>*** options with the formula:

$$r_i = \frac{p_i}{t_i} \times c_1 + \frac{w_i}{t_i} \times c_2 + \frac{s_i}{t_i} \times c_3 \ (1 \leq i \leq n)$$

Here *n* is the number of elements in the corresponding database; p_i, w_i and s_i are the factors of *i*th triplet of the database and $t_i = p_i + w_i + s_i$.

The combination B$_i$ which has the minimum r_i is the selected combination of optimization options. In this way, the goal to have a trade-off on the optimized combination of optimization options is achieved.

The implementation of our trade-off strategy does not involve significant changes to the original compiler, nor does affect compilation time.

4.3. Testing Example

After implementing the trade-off strategy and integrating the set of databases generated from the DSP measurement suite in KCC, we use the remaining 4 DSP kernels to test our strategy.

First, we use the *P*, *W* and *S* options individualy to test each factor separately. The combinations automizatically selected by KCC are B1 for *P*, B17 for *S* and B18 for *W*, respectively.

Figure 5 shows the optimized results of the two factors of performance and code size. The reason why there is no obvious difference for the power factor is that the small scale of the DSP measurement suite.

After then, we test with weighted coefficients 0.2, 0.3 and 0.5 for *P*, *W* and *S* factors, respectively. The selected combinations of KCC is B12 after the trade-off. The testing results are also shown in Figure 5.

For all the test cases we observe significant improvements in performance and code size, except *fir*, for which execution time increases compared to the baseline.

5. Related work

The research of exploring the optimal combination of compiler optimization options to improve the performance, lower the power consumption, and reduce the code size for embedded systems is still in its infancy. There are few studies in the literature focused on this problem.

In [8] reported a current study of optimal using the compiler options to improve the performance of applications.

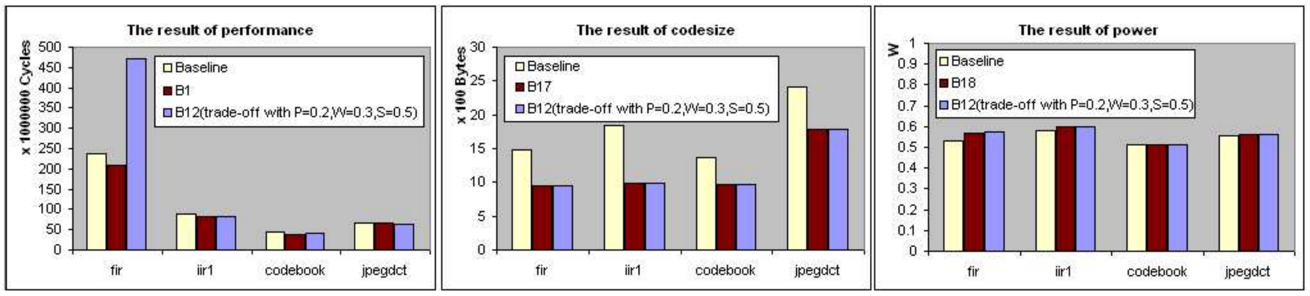

Figure 5. Test result

Another heuristic compiler option recommendation is developed to deterministically select PA-RISC compiler options, based on the information provided by the user, the compiler and the profiler [5]. In another work, various compiler optimizations are applied concurrently and the resulting energy consumption is evaluated via simulation [9].

The above studies only consider one specific factor, especially the performance, or discuss the general issues of why and how a combination of compiler optimization options affect the performance improvement or power dissipation. We are not aware of compiler support. From the view of the application developers, they would be responsible to search for the optimal combination of compiler options if no compiler support is provided.

In contrast, we present the *UMECO* methodology to integrate the three primary factors of performance, power and code size as a whole and the compiler creates an optimized combination of compiler options automatically to achieve the specific goals of the application developers. In this way, the users only need to describe their requirements to the compiler without knowing any of compiler options.

6. Conclusion

We present a practical methodology of automatic exploring compiler options(*UMECO*) with which applications can achieve optimized results in terms of performance, power consumption or code size, even when application developers have limited knowledge of the optimization techniques deployed by the underlying compiler.

We describe the *UMECO* methodology, implement it on a real architecture, the Intel XScale microarchitecture, and test it with a suit of benchmarks.

The work reported in this paper is still in progress. Because of the limitation of the 80200 board, we can not run bigger benchmarks, such as Mediabench, Mibench, etc. We intend to apply the *UMECO* methodology on another platform, like the Intel XScale PXA250.

Acknowledgments

We wish to acknowledge support from DOD, DOE (Award No. DE-FC02-01ER25503) and NSF (awards No. CCF-0541002 and CNS-0509332). We acknowledge Ziang Hu, Shuxing Yang, Hongbo Rong and Sun Chen for their support in the development of the KCC compiler infrastructure, which was used in our experiments. We wish to acknowledge Joseph B. Manzano for a thorough review of the final manuscript.

References

[1] *Kylin C Compiler*. http://www.capsl.udel.edu/kylin.

[2] Wasabi@ software Development Tools User's Guide for Intel Xscale Microarchitecture, WASABI Systems, Inc, March 2004.

[3] M. Alt. *Performance Modeling Using Compilers*. Intel Corporation, May 2005.

[4] E. Ozer, A. P. Nisbet and D. Gregg. Classifi cation of Compiler Optimizations for High Performance, Small Area and Low Power in FPGAs. Technical Report. Department of Computer Science, Trinity College, Dublin, Ireland, June 2003.

[5] E. Granston and A. Holler. Automatic recommendation of compiler options. In *Proceedings 4th Feedback Directed Optimization Workshop*, Dec. 2001.

[6] H. P. Wu, L. Chen, J. Manzano and G. R. Gao. A user-friendly methodology for automatic exploration of compiler options. In *The 2006 International Conference on Programming Languages and Compilers(PLC'06)*, Las Vegas, USA, June 2006.

[7] L. Kane. Creating high performance embedded applications through compiler optimizations. In *Technology @Intel Magazine*, Intel Corporation, March 2005.

[8] M. Haneda, P.M.W. Knijnenburg and H.A.G. Wijshoff. Optimizing general purpose compiler optimization. In *CF'05*, Ishia, Italy, May 2005.

[9] M. T. Kandemir, N. Vijaykrishnan, M. J. Irwin, and W. Ye. Influence of compiler optimizations on system power. In *Design Automation Conference*, Los Angeles, California, 2000.