

Implementing Click IP Router Kernel on VLIW Architectures

Xiaodong Jin and Kanyu Cao
CS252 Class Project, UCB

Abstract

In this work, we implemented the Click IP Router Kernel in C language provided by Scott Webber et al. for two VLIW processors designed for DSP purpose, namely the Philips Trimedia TM1300 processor and Texas Instrument TMS320C6701 processor. The performance of these processors are compared with those of three other processors, ARM SA-110, HPL-PD EPIC, and Intel IXP1200 [1]. Ways of further performance optimization is explored on both processors by using techniques such as function inlining, loop unrolling, and grafting.

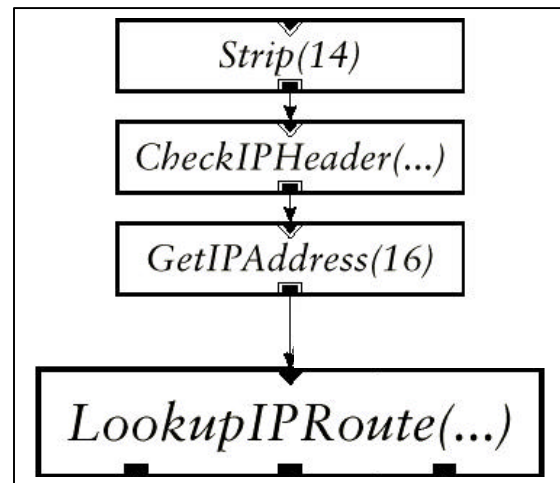
Introduction

The very long instruction word (VLIW) architecture is considered to be one of the promising methods of increasing performance beyond standard RISC architectures. Although RISC architectures take advantage of temporal parallelism (by using pipelined functional units), VLIW architectures can also take advantage of spatial parallelism by using multiple functional units to execution several operations concurrently. Similar to superscalar architecture, the VLIW architecture can reduce the clock per instruction (CPI) factor by executing several operations concurrently. Compare to superscalar architecture, VLIW has the advantage of simplified hardware for decoding and issuing instructions. Since it is easier to exploit the instruction level parallelism (ILP) in the stream, it is currently found more successful in the DSP applications. Networking protocol application is also one of the applications where the ILP can be explored. In this study, we will explore the performance of the two VLIW architectures: the latest Philips Trimedia and TI TMS320C6701 processors, by implementing an IP routing kernel. The results are also compared with other architectures. It shows that the VLIW architecture has the performance advantage over the standard RISC architecture. However, it is not as good as dedicated networking protocol processor such as Intel's IXP1200. The following IP address annotation is checked and the destination gateway is write into the destination annotation and. Then the packet is send to the

section will briefly introduce the Click IP router program we implemented. Then the performance and optimization on both Trimedia TM1300 and TI TMS320C6701 will be described in detail.

Click IP Router Kernel

Click is a modular software router developed by Parallel and Distributed Operating System group of MIT. Its main advantage is in its modular structure and standard language implementation. These features greatly simplify the configuration process and make the router scalable. In this



project, only a very simple kernel of an IP router is implemented as shown in Fig. 1.

The first element, which is the basic building block in Click, is used to strip off the useless information, such as Ethernet header, from the

Fig. 1 Illustration of the Click router elements

beginning of the packet. Then the stripped packet is passed on to the CheckIPHeader element. This element will check the validity of the IP header. Invalid packets are dropped if no disposal port is used. Then the following element GetIPAddress will copy the IP address of this packet to the destination IP address annotation for use by other elements. The final stage of the kernel is the LookupIPRoute stage, where the destination

right port corresponding to the gateway. A separate random traffic generator program

generates the packets used for performance evaluation.

Philips Trimedia TM1300

1. Introduction

Trimedia TM1300 [3] is a high performance very long instruction word (VLIW) processor specially optimized for communication or multimedia signal applications. It can issue at most 5 instructions at the same time. The maximum clock frequency is 166MHz. It has 128 32-bit general-purpose registers and on-chip hardware support for audio-video I/O. It also provides many C/C++ callable special operations to speedup SIMD (single instruction, multiple data) operations common in DSP applications. Philips also developed a powerful software development environment SDE2.1, which has an ANSI-compliant C/C++ optimizing compiler, a cycle-accurate, machine level simulator, a source level debugger, and powerful profiling and performance analysis tools. It also supports profile driven compilation.

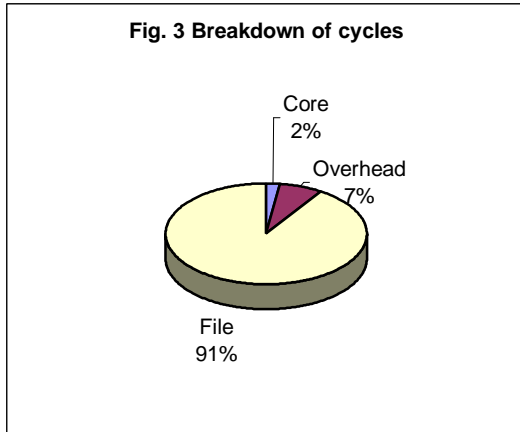
2. Implementation and Optimization

We implemented the original Click IP Router program by compiling it with the compiler using several options. One case is the standard options

of the compiler and scheduler, the second case is to turn the memory delay model off and assume perfect memory system, and the third case is to use the recompile function of the compiler. The last case is a unique feature provided by the SDE2.1 to let the compiler take the advantage of the real execution profile by recompiling after executing and profiling. Function calls of each case is very carefully profiled and analyzed. For example, Figure 2 tabulates the function call profile by cycles consumed for the default compiler options. The functions related to memory access are identified and put into red color. Considering the fact that in real routers traffic data are streamed in instead of stored in files, we removed the contribution of these functions from the execution time to get the maximum performance. The functions that are executed only finite number of time due to program initialization are characterized by blue color and are also deducted from the final performance number. The number of “core function” cycles is only a small portion of the total cycles executed. A breakdown of the contributions is illustrated in Fig. 3. The same procedure is carried out for the cases of perfect memory and recompilation. The results are summarized in Fig. 4.

Function	Executions	Total Cycles	(%)	I\$ Cycles	D\$ Cycles	File	Total Cycles
_____	_____	_____	_____	_____	_____	_____	_____
__svfscanf	85012	16392	41.43	2	59	__svfscanf	16392
__strtoul	85042	8447	21.35	1	3	__strtoul	8447
_rt_umod	85042	3742	9.46	0	0	_rt_umod	3742
_rt_udiv	85042	3402	8.6	0	0	_rt_udiv	3402
_InfiniteSource_init	1	2000	5.06	1	22	_AppSem_V	1531
_AppSem_V	85025	1531	3.87	0	0	_AppSem_P	1361
_AppSem_P	85025	1361	3.44	0	0	_fscanf	681
_fscanf	85012	681	1.72	0	1	__strtoul	336
__strtoul	84041	336	0.85	0	0	total	35892
__malloc	4087	328	0.83	2	28	Overhead/Idle	
_RTM_Init	1	207	0.52	1	139	_InfiniteSource_init	2000
_memset	16	194	0.49	0	148	__malloc	328
_Discard_push	1000	107	0.27	0	0	_RTM_Init	207
_memcpy	1000	79	0.2	0	40	_memset	194
_LookupIPRoute_push	1000	69	0.17	0	0	_InfiniteSource_push	36
_in_cksum	1000	62	0.16	0	0	_dummy_fun	33
_CheckIPHeader_simple_act	1000	59	0.15	0	0	total	2798
_Element_push	2000	52	0.13	0	2		
_InfiniteSource_push	1	36	0.09	0	3	Core	874.232

Fig. 2 Number of cycles and executions for function calls for standard compilation.



The processor with perfect memory performs the best and its number can be considered the best possible and is about 220K packets/Sec. Also from the figure, we can conclude that for our application, recompilation degrades the overall performance from 190K packets/Sec to 181K packets/Sec.

3. Optimizations

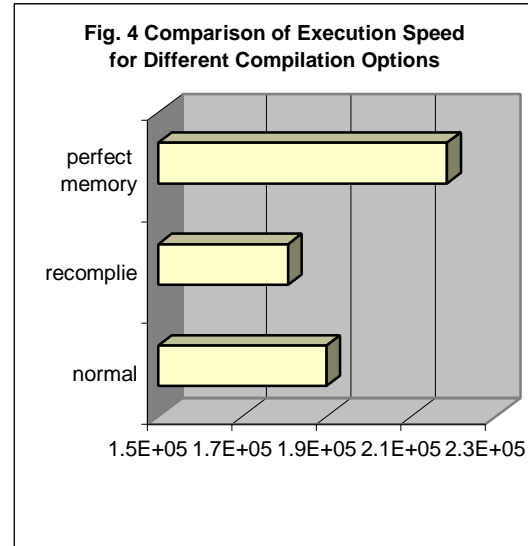
Although only the kernel of an IP router is implemented, it is still too big to be optimized at the instruction level for this project. So we will study the optimization in two ways: one is to explore the effect of using different optimization options during compilation. The second is to optimize the most cycle-consuming core function "discard_push" (shown in Fig. 2).

Fig. 5 listed the number of core cycles and code size when different level of optimization options

option	core cycles	code size
normal	874.232	230K
O5	819.5	221K
O5g	759.5	232K
rO5g	759.5	232K

Fig. 5 Effects of optimization options

are used in compilation. The normal option is the default options of the compiler. At this level, the compiler performs local/global optimization and optimizes variables in registers. At O5 level, it does increased inter-procedure global optimization and function inlining. In case of O5g, the compiler use grafting on top of the O5 level. In the last case, rO5g, a recompilation is performed. From the figure we can see that



global optimization/function inlining and grafting boost the performance by about 7% each. On the other hand, as discovered in the previous section, recompilation does not help in this particular application.

We also studied how to optimize the function discard_push. The code is shown in Fig. 6.

From Fig. 6, we can see that it contains a loop

```
#include "crack.h"
#include "discard.h"
#include "packet.h"
unsigned int out_channel[21];
void Discard_push(Element *_this, int port, Packet *p) {
    int i;
    ClickIP *ip = (ClickIP *) Packet_data(p);
    unsigned int *ptr = (unsigned int *) ip;
    #pragma TCS_unroll=21
    for(i = 0; i < 21; i++) {
        out_channel[i] = ptr[i];
    }
}
```

Fig. 6 Code for function discard_push

with a fixed iteration number, so loop unrolling should help here. So we added the line started with "#pragma" just before the loop to manually unroll it. Fig. 7 shows the effect of loop unrolling on the cycles consumed by this function.

We also tried some other approaches to improve the performance. For example, TM1300's cache

option	Cycle
normal	107
manual loop unrolling	88

Fig. 7 Effect of loop unrolling on discard_push

access has a granularity of 64 bytes and is aligned at 64-byte boundaries in memory. So if data is not aligned, there is a chance to increase memory access. We tried to use the custom cache_malloc function instead of the standard malloc function to align our data to the 64-byte boundary, but that caused check-sum problem for the router program.

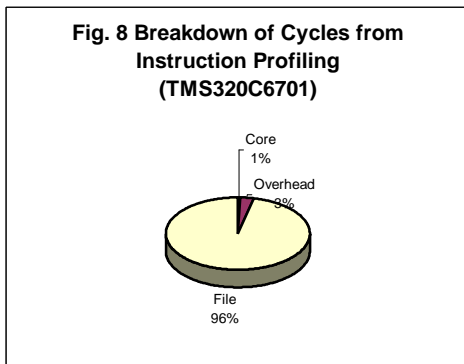
TI TMS320C6701

1. Introduction

TI's TMS320C6701 is also a VLIW architecture offering 1336 MIPS and 1GFLOPS at 167MHz. It can execute up to 8 instructions in single cycle (6 float, 2 integer). It has 128K on-chip RAM and thirty two 32-bit registers. The internal memory has two blocks: 64Kbytes for program memory or cache and 64Kbytes for internal data memory. The internal program memory is organized as 2048 x 256-bit instruction packets. It may be used directly as program memory, or as cache memory for code running in external memory. In both cases, only complete packets, (256-bits), are fetched, and this is done in a single cycle. Both register files may access the 64KByte internal data memory simultaneously. It also has a C/C++ compiler. The simulator and compiler of this processor can be download from TI's website.

2. Implementation and performance.

We also implemented the click router kernel in TI's TMS320C6701 [4]. The simulator provides a profiling function by which we can know the cycles for executing certain code section. The result is shown in Fig.8. It shows almost 96% of the time is used to read and write to files that are an artifact of our implement approach, since we only implement part of the router. The core function only takes 1% of the time. If we



translate the 1% core cycles to forwarding speed according to the clock frequency, it can forward **140K** 64-byte packets per second. There is a compile option for tradeoff the code size for performance. But it only change the code size by less than 1% if we using the highest speed option. So we always use this option to compile our code.

Comparison

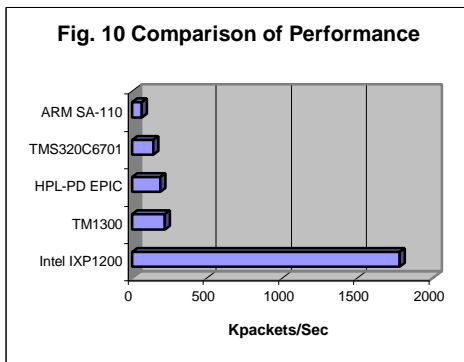
The comparison of the two processors is listed in

	Trimedia TM1300	TI TMS320C6701
Clock Speed	166MHz	167MHz
Slot Wide	5	8
Delay Slots	3	5
Design Effort	3 days	5 days
Code Size	230kB	128kB
Routing Speed (Perfect Memory)	2.19×10^5 Packets/Sec	1.4×10^5 Packets/Sec

Fig. 9 Comparison of TM1300 and TMS320C6701

Fig.9. They have almost same operating frequency. Although TM1300 has smaller issue width (5 vs. 8), it has less delay slot (3 vs. 5). In our application, there are lots of branches, so it helps it run faster than TMS320C6701. Other possible reason could be that it has more registers than TMS320C6701. However, for the same original C code, TMS320C6701 generate much small size of executable code than what TM1300 does. The GUI of TI's code composer is more friendly than Trimedia's SDE. However, it is more complicated so it takes more time to complete our implementation. It is also possible that we didn't find the optimum compilation, so we can't further improve its speed just like what we do with TM1300.

Scott Weber et al. [1] compared the performance of the same program on three other processors, ARM SA-110, HPL-PD EPIC, and Intel IXP1200. Fig. 10 compares their result on these processors with our result. It shows that the VLIW architectures show several times faster than the standard RISC processor (StrongARM) because it exploits the instruction level parallelism. However, it still can't catch the speed of the dedicated network protocol processor (IXP1200), which has multi-threads function. The IXP1200 are about 10 times faster than the VLIW processors.



Conclusion

In this work, we implement the kernel of the Click IP router on two VLIW architecture DSP processors: Philip's Trimedia and TI's TMS320C6701. The optimization of the implementation is also explored. The performances are compared between these two processors as well as with other architectures. It shows the VLIW do show performance advantage over standard RISC processor (StrongARM), but cannot catch the dedicated network protocol processor (IXP1200). The Trimedia is a little faster than TI's processor for this particular application. It may because Trimedia has more registers and less delay slots. However, the TMS320C6701 has much smaller code size.

References

- [1] Scott Weber and Fernando De Bernardinis, Exploration of the new IXP1200 Network Processor from Intel. CS252 Final Project, Fall 1999.
- [2] Parallel & Distributed Operating System Group, MIT, The Click IP Router Project. <http://www.pdos.lcs.mit.edu/click/>.
- [3] More detailed information on Trimedia processors is available at <http://www-us2.semiconductors.philips.com/trimedia/>.
- [4] More detailed information on Trimedia processors is available at <http://www.ti.com/sc/docs/products/dsp/tms320c6701.html>.