Performance of Embedded System Application on Network Processor

---Directed Study Project Report

Danhua Guo
Department of Computer Science and Engineering
University of California, Riverside
dguo@cs.ucr.edu

Abstract

As the market of Network Processor (NP) has become more and more intriguing recently, a lot of research on architectural level has proved that NP's performance on packet processing is quite satisfactory. The further look at NP's architecture reminds people of that of embedded processors. So an interesting question is: How will NP perform with embedded system application? In this project, I run Mibench benchmark on Intel IXA 2400 NP, and compare the result with different multithreading and multi-microengine settings. I found that the performance of NP is also very impressive with embedded applications.

1 Introduction

Recently, Network Processor (NP) has become a hot topic in the field of computer architecture and networks. The motivation of NP is to share the advantages of General Purpose Register (GPR) and Application Specific Integrated Circuits (ASIC), namely to have a low-cost, high-performance processor with programmable flexibility.

Many research and study have provided positive evaluations of NP on a wide range of applications, such as packets classification in the router or switch (filtering/forwarding), traffic management (queuing, scheduling and policing packet data), and control processing (macro level control of packet operation), etc. However, not enough concern has been drawn in the field of embedded systems, even though the market of which has become more and more popular and the structure of embedded processor (ARM + DSPs) is quite similar to that of NP (XScale + MEs). Mostly, when it comes to embedded applications, the balance of device performance and power consumption is always the key issue. Like PDA and cell phone communication, users keep complaining about the battery life along with the response latency. With the help of NP embedded in such portable devices, it is expected to have a better solution to this problem.

In this project, I test the embedded features of NP with the help of MiBench benchmarks, which is targeted particularly at embedded features. The first step is to use Intel's Internet eXchange Architecture Software Development Kit (Intel IXA SDK) as the basic simulator to get some explicit result and therefore verify NP's relevant features. After that, there is a study into the detailed impact of different configuration of NP components on its performances.

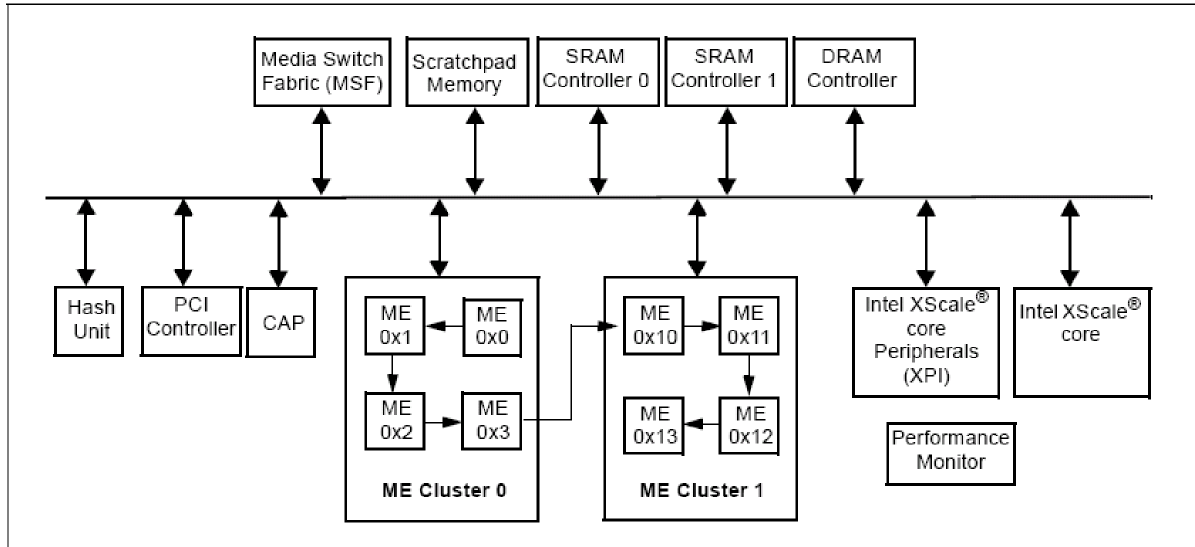
The report will be organized as follows. In section 2, a brief introduction of the architecture of Intel IXA NP2400 will be given. Plus, two software programming models will also be discussed here. In section 3, we described the benchmarks and simulators in detail. Section 4 will present the result from the

simulation, analysis and comparison between different sources will also be provided. In section 5, we conclude the paper with future extension related to this paper.

2 Architecture of Intel IXA NP2400

2.1 Block Diagram and Key Features

Figure 1. IXP2400 Chassis Concept Block Diagram

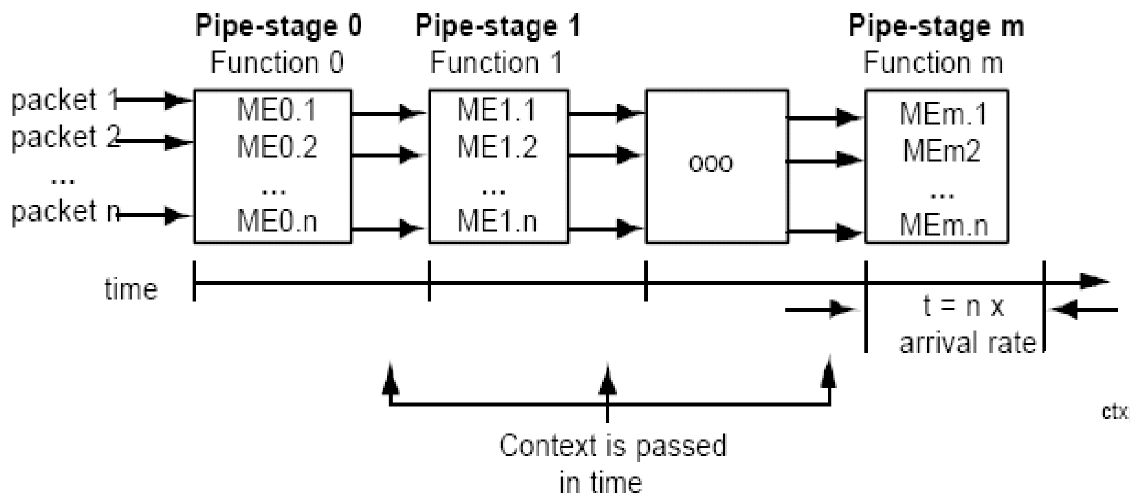


IXP 2400 Resource Summary

Name	Size (Bytes)	Transfer Size (Bytes)	Reference Latency (Cycles)	Application
XScale Core	32-bit general-purpose processor			Initialize and manage the chip
GPR	256*4	4	1	General programming purpose
XferR	512*4	4	1	Transfer data to and from MEs
NNR	128*4	4	1	Transfer data from previous/next neighbor ME
LM	640*4	4	3	Caching data needed by ME
Scratch	16K	4	60	General purpose use with atomic operations and ring support
SRAM	64M	4	90	Control information storage
SDRAM	2G	16	120	Data buffer storage

2.2 Programming models

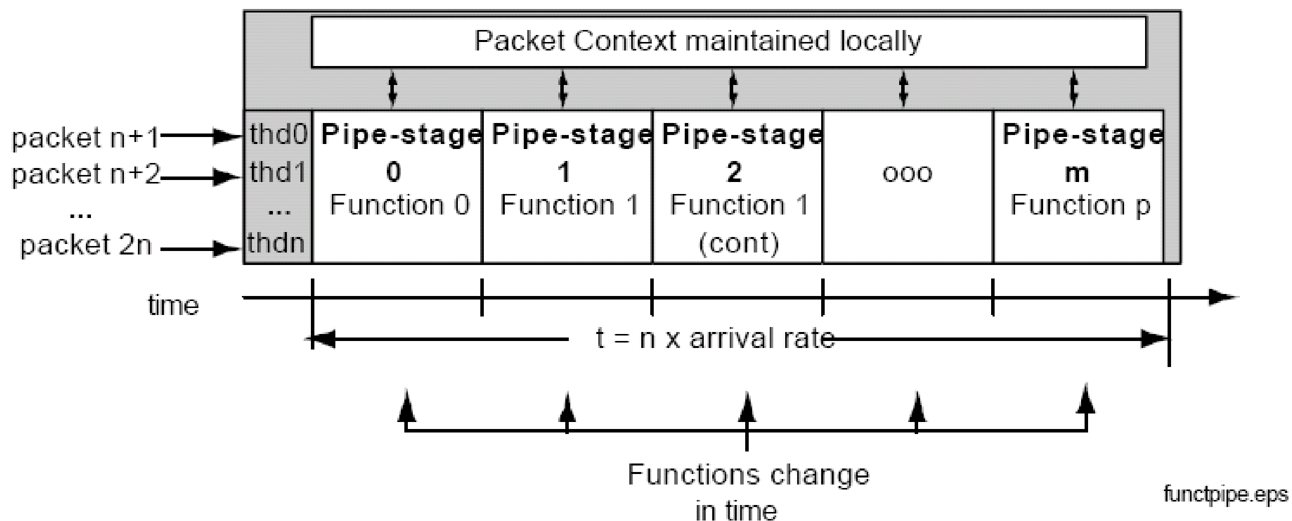
2.2.1 Context pipeline



The advantage of the context pipeline is that the entire ME program memory space can be dedicated to a single function. This is important when a function supports many variations that result in a large program memory footprint. The context pipeline is also desirable when a pipe stage needs to maintain state (bit vectors, or tables) to perform its work. The context pipe stage can use the local memory to store this state eliminating the latency of accessing external memory.

Cases where the context pipeline is not desirable are ones in which the amount of context passed to and from the pipe stage is so large that it affects system performance. Another disadvantage of the context pipe stage is that all pipe stages must execute at minimum packet arrival rates. This may make partitioning the application into stages more difficult.

2.2.2 Functional pipeline



In a functional pipeline, the context remains with an ME while different functions are performed on the packet as the time progresses. The ME execution time is divided into "n" pipe-stages and each pipe-

stage performs a different function. A single ME can constitute a functional pipeline. The functional pipeline gets its name from the fact that it is the function that moves through the pipeline.

Packets are assigned to the ME threads in strict order, so that if there were "n" threads executing on an ME, the first thread, "A" must complete processing its first packet before "n" + 1st packet arrives so that it can begin processing the "n" + 1 packet.

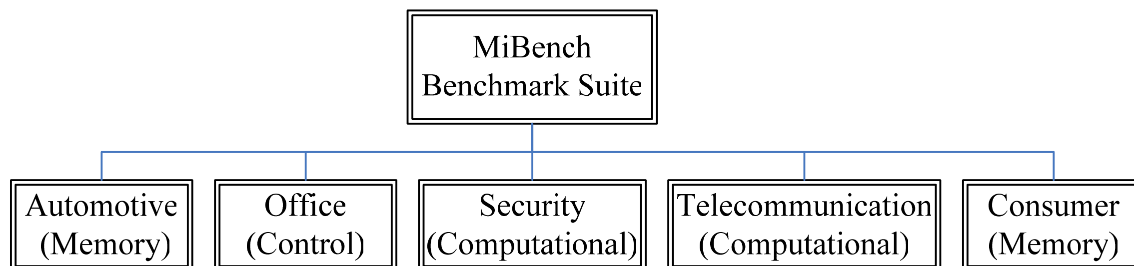
2.2.3 Comparison between Context Pipeline and Functional Pipeline

	Context	Functional
Advantages	1. Good for programs with a large code size 2. Good for on chip storage of vectors/tables	1. Supports a longer execution period than context pipe-stages 2. Exe time for each pipe-stage is flexible
Disadvantages	1. Bad for large context passing 2. All pipe stages must execute at minimum packet arrival rates. (hard to partition)	1. ME must support multiple functions 2. Mutual exclusion may be more difficult

3 Benchmarks and Simulators

3.1 Introduction to MiBench

MiBench is a benchmark suite developed by University of Michigan. The motivation of their project is to provide an open source benchmark for embedded applications. The figure below shows how the benchmark is organized.



The next table shows the instruction distribution information of the benchmark suite.

Different packages	Integer/floating	Memory	control	Function
Telecommunication	50%++			Find and generate entropy by repeatedly operate on a datum
Security	50%++			
Network				

Consumer		++		Large image data is processed
Office, Automation		++	++	Function calls to string library to manipulate ASCII data (lots of branches)
SPEC	57%++	57%	57%	
ADPCM encode/decode	80%+(integer)			Part of Telecommunication

++: the package has a strong attribute of certain category.

Embedded system application is always evaluated in three aspects. Firstly is a control intensive application, such as branch instructions where many control flow is concerned. The second group is computational intensive application, like integer and floating point ALU operations. The last group involves I/O intensive applications. E.g. memory (load and store). In my project, I select one application from Office package, which is a control intensive application and another one from security, which has more computational operations.

3.2 Introduction to Intel IXP SDK

Intel IXP SDK is provided to simulate its network processor and evaluate the result with specific component configuration, such as the number of micro-engines and the context mode (4, 8) as well as the number of threading running in each micro-engine.

However, one of the things that are not convenient for the user is that SDK only supports limited sources of input stream. They call it MicroC or MicroCode, depending on the level of language similarity to assembly. The reason for the existence of such "mid-ware" language in my opinion is 1) NP needs explicit configuration, therefore requires the compiler to figure out which value to be distribute to each component; 2) the new compiler doesn't fully implement all the ASCII C library. A list of the ASCII C library that is not supported by Intel C Compiler is provided in the table below. As a result, the input should be specified valid to the compiler.
A list of unsupported ASCII C attributes:

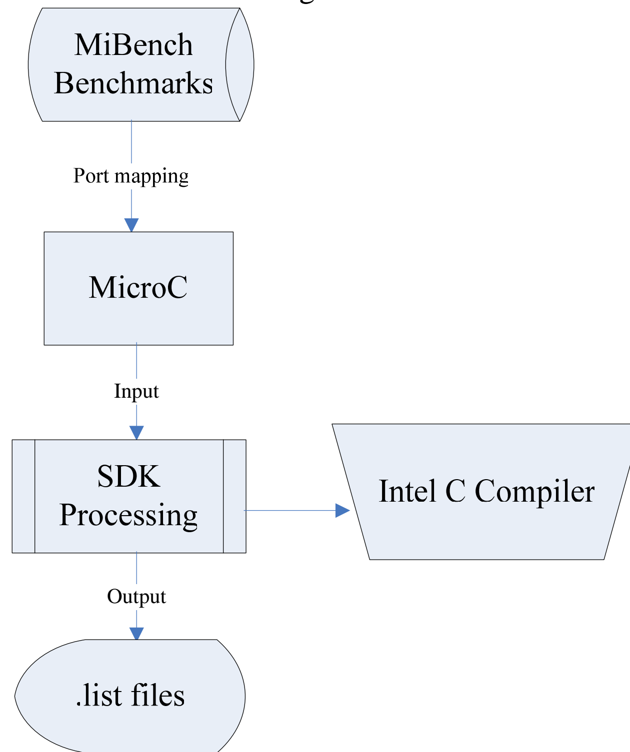
1. The full standard C runtime library.
2. Automatic parallelization of code.
3. C++
4. Floating point data types (float and double).
5. Function pointers and recursion.
6. Functions with a variable number of arguments (varargs).

Unfortunately, most of the input benchmarks are written in standard C. Hence a preprocessing port-mapping is required before it is taken by NP. More detail about port-mapping will be provided in the next section.

4 Performance Result

4.1 Step of the experiment

Below is the flow diagram of the experiment. As we can see, before we run the benchmark in SDK, it is required to be port-mapped into MicroC first. Then with the help of SDK, we could get the .list file and then feed it back to SDK or other NP simulator to get the result.



The tricky part in this project is how to port map the original C program into MicroC. First we need to consider Data Allocation, namely where to store the data. Remember we could store a variable in a register (GPR, NN, Xfer), or we could use Memory (LM, Scratch, SRAM, DRAM). As long as we figure out which one of the regions to allocate for the data, the keyword __declspecs will help to do the job.

Now the problem is how to decide which region to use. The answer is to draw a balance between the size of the data and the requirement of accessing speed of it. Usually the smaller a region is, the shorter time it takes to read and store a data from it. In Intel C Compiler, if one of the registers or memory is used up, but the user write explicitly in the code to allocate data to the same place, a “register spillage” will happen. And the compiler will automatically allocate the next fastest register or memory to that data. For example, a variable will be automatically allocated in the following order: GPR -> NN -> Local Memory -> SRAM so on and so forth.

After we port map the benchmark into the compiler-friendly version, it’s time to configure the SDK with component specific information. First it is required to tell the compiler which library to include for the project and where to locate the library. This is important if some system functions are used. It is also possible to reserve memory locations for certain variables. Meanwhile, the number of ME to use and context model for each ME could also be set up in SDK. By setting up a microengine to run in 4-

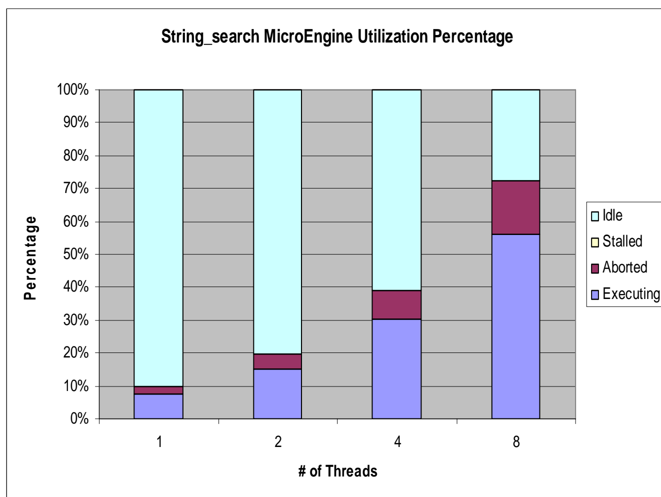
context mode, each context can access twice as many context relative registers. In this mode, odd contexts 1, 3, 5, and 7 are disabled and the even contexts 0, 2, 4, and 6 have full access to their registers.

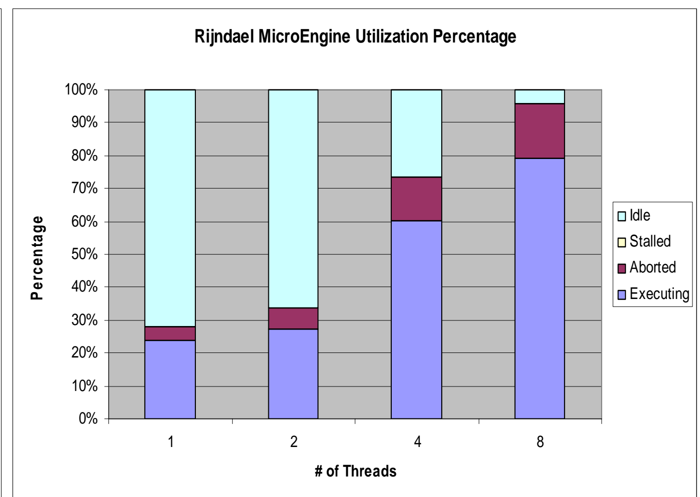
4.2 Result

I did my project with the following environment set up.

- Intel IXP SDK 4.1 (Select IXP2400)
- 600MHz ME configurations
- 200-MHz SRAMs
- 150-MHz RDRAMs
- Executed in Multi-threads
- Executed in Multi-MicroEngines

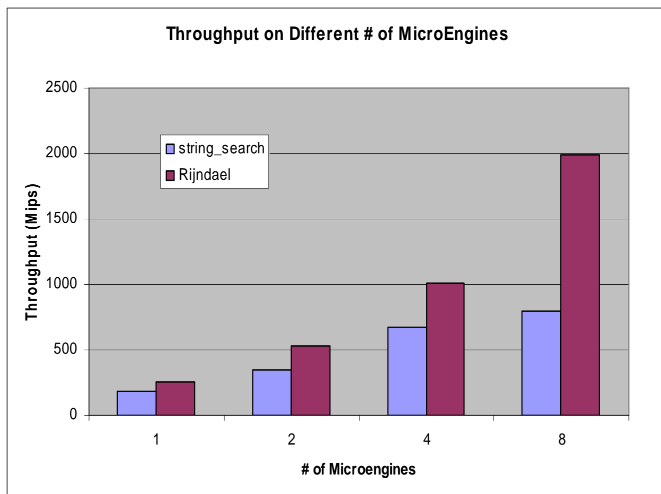
4.2.1 Micro-engine Utilization Percentage (On 8-Context Mode on 1 ME)



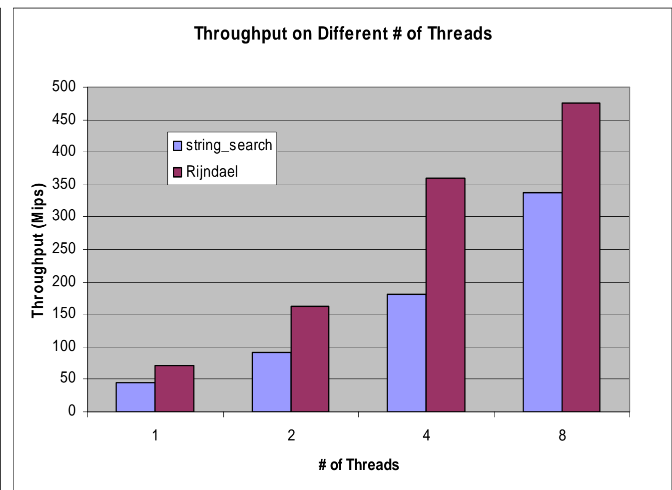


4.2.2 Throughputs (Mips)

Multi-Core



Multi-threading



5 Conclusion

As we can see from the Micro-engine utilization diagram, both string_search and AES application have an increase on the percentage of execution time on each ME with the increment of the # of threads.

In multi-core diagram, changing the number of MEs is based on fixed number of threads. In my experiment, it's on 4-Context Mode with 4 threads. In multi-threading diagram, changing the number of Threads is based on fixed number of MEs. In our experiment, it's on 8-Context Mode with 1ME.

The reason why we select 4-Context Mode is because with the increasing number of ME, the competition for memory is fiercer. So we reduce the number of threads on each ME to make sure all the data are allocated to valid slots in the limited memory.

We can see that with multi-threading and multi-core, the throughputs of NP increase almost in a linear fashion. Therefore we can conclude that embedded system application should be efficient to run with a NP and the performance is also guaranteed to be satisfactory.

6 Reference

[1] Intel C Compiler User's Guide
[2] Intel IXA Development Tools User's Guide
[3] Matthew R. Guthaus et. al., *"MiBench: A free, commercially representative embedded benchmark suite"*
[4] Zhangxi Tan et. al., *"Optimization and Benchmark of Cryptographic Algorithms on Network Processor", 2004*
[5] Intel Corporation – Intel® IXP2400 Network Processor - 2nd Generation Intel® NPU
[6] Yan Luo et. al., *"NePSim: A Network Processor Simulator with a Power Evaluation Framework", 2004*
[7] Dominic Herity, *"Network Processor Programming", http://www.embedded.com/story/OEG20010730S0053*