Performance of Embedded System Application on Network Processor

2006 Spring Directed Study Project
Danhua Guo
University of California, Riverside
dguo@cs.ucr.edu
06-07-2006

Motivation

- NP Overview
  - Programmability (GPP)
  - Performance (ASIC)
- Project Motivation
  - Architectural Portability

Motivation (Cont.)

- NP has been proved to be a successful design for packet processing.
- NP architecture (XScale + MEs) is similar to Embedded Processor architecture (ARM + DSP).
- Embedded application performance on NP?

Road Map

- Motivation
- Intel IXA NP: A Quick Look
- Environment: Intel IXA SDK
- MiBench Overview
- Experiment

Intel IXA 2400 NP Architecture

Environment: Intel IXA SDK

MiBench Overview

Experiment
### IXA 2400 NP Resources Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Size (Bytes)</th>
<th>Transfer Size (Bytes)</th>
<th>Reference (Cache)</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data buffer</td>
<td>120</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>General</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scratch</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Caching</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transfer</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hold</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context</td>
<td>32-bit</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Programming Model

- **Context Pipeline**
  - Context is passed through the pipeline
  - A single ME is dedicated entirely to a single function

- **Context Vs. Functional**
  - +++: Good for programs with a large code size
  - Good for on-chip storage of vectors/tables
  - Exe time for each pipe-stage is flexible
  - ME must support multiple functions
  - Mutual exclusion may be more difficult

- **Which one is better?**
  - A combination of Context and Functional Model

### Road Map

- Motivation
- Intel IXA NP: A Quick Look
- Environment: Intel IXA SDK
- MiBench Overview
- Experiment
**Environment:**

**Intel IXA SDK Workbench**
- A software development kit for NP programming
- Input: MicroC or MicroCode
- Configuration: Data Flow, ME distribution, etc.
- Output: .list file for each ME

**Road Map**
- Motivation
- Intel IXA NP: A Quick Look
- Environment: Intel IXA SDK
- MiBench Overview
- Experiment

**MiBench Overview**
- Type of Embedded Application
  - Control Intensive: branch instructions
  - Computational Intensive: integer and floating point ALU operations
  - I/O Intensive: memory (load and store)

**Road Map**
- Motivation
- Intel IXA NP: A Quick Look
- Environment: Intel IXA SDK
- MiBench Overview
- Experiment

**Experiment Outline**
- Port map MiBench to SDK
- Configure SDK Environment
- Feed .list into SDK
- Analyze and Evaluate Result

**Port mapping**
- Why is it required?
  - The C compiler is not a complete ANSI C implementation.
  - Things the compiler does not support
    - the full standard C runtime library.
    - automatic parallelization of code.
    - C++
    - floating point data types (float and double).
    - function pointers and recursion.
    - functions with a variable number of arguments (varargs).
Port mapping (Cont.)

- How is it working?
  - Data Allocation
    - Register Regions: GPR, NN, Xfer
    - Memory Regions: LM, Scratch, SRAM, DRAM
    - Allocation attributes (declspecs) can describe where
      the variable is allocated.
  - Register Spillage
    - Default spill order:
      - Next Neighbor registers
      - Local Memory
      - SRAM Memory
  - Functions
    - Inline function (function calling overhead)
    - Optimizing pointer arguments (callee access overhead)

Optimizing Your Code

- Minimize variable allocation to memory.
  - Taking the address of a variable or declaring it with a
    memory region attribute causes allocation to memory.
  - Structures/arrays larger than 64 bytes (or 128 bytes in 4- context mode) are allocated to memory.
  - Minimize access to memory variables.
  - When possible, declare a variable as a local variable in
    main() rather than as a global or a static variable so that
    unnecessary initialization is avoided.
  - Efficient Structure Access
    - Sizing of Structure Members
      - A multiple of four bytes in size
      - Falling on a four byte offset from the start of a structure
      - Do not cross a 4 byte boundary

SDK Configuration

- Library directory
  - Which library to include
- Memory information
  - Which part of memory segments is reserved for
    variables
- Thread information
  - Which mode to use
    - # of contexts (threads) to run
- ME information
  - Which .list file to be fed into which ME

Environment

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

Experiment Result

- MicroEngine Utilization Percentage
  - On 8-Context Mode on 1 ME

Experiment Result (Cont.)

- Throughput (Mips)
  - Multi-core
  - Multi-thread
References

- Intel C Compiler User's Guide
- Intel IXA Development Tools User's Guide
- Matthew R. Guthaus et. al., "MiBench: A free, commercially representative embedded benchmark suite"
- Intel Corporation - Intel® IXP2400 Network Processor - 2nd Generation Intel® NPU

Thanks to

- Prof. Bhuyan for the idea of simulation
- Prof. Luo for the suggestions on SDK
- Mr. Piyush Satapathy for the experience on MicroC coding
- Everyone for giving up your valuable “afternoon-coffee” time

Questions or Comments?