CS229 Final Project

Title: Fault-Tolerant Architecture for Machine Learning Applications

Larkhoon Leem (SUID: 5149427)

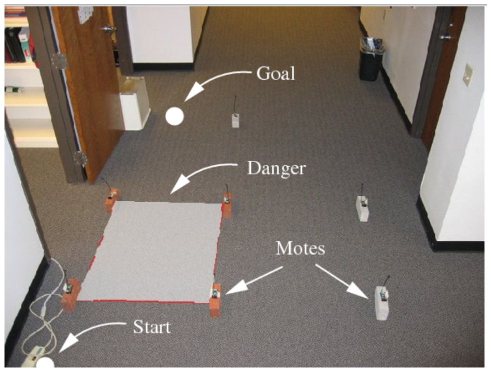
1. Introduction

Scaling of semiconductor transistors in integrated circuits has been major thrust for speed and density improvement of modern microprocessors. At the same time, ever decreasing semiconductor size makes it more difficult and expensive to make microprocessors strictly reliable.

If we look around it is not so hard to find systems that allow errors. Those systems take advantage of error-detection and error-correction mechanism to live up with errors. An example of systems is telecommunication. Noise is inevitable in data transmission. Efficient encoding and decoding schemes have been invented to fight with noise errors. Encoding and decoding schemes use iterative and probabilistic algorithms. Iterative characteristics provide good error-detection and error-correction points. Probabilistic nature of output result allows relaxed requirement for accuracy of computation.

Machine learning applications shares many similarities with telecommunication decoding/encoding. Estimation application and categorization algorithms are iterative and probabilistic. If we can take advantage of these characteristics of machine learning we can build an error-resilient computer architecture which is cost-effective in terms of testing and power consumption.

In this project report I would like to propose new way of computing for fault-tolerant machine learning application. Background information will be given in section 2. Section 3 will explain experiments which have been conducted in this project. In section 4, architectural design proposal based on experiments will be given.



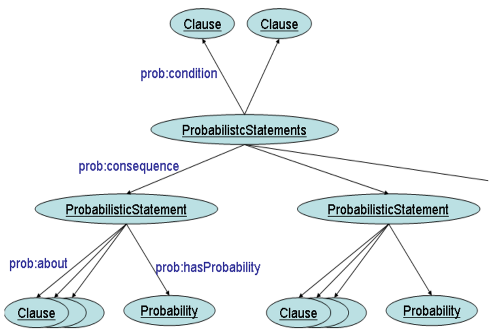


Figure 1 Machine learning & probabilistic applications

2. Fault-tolerant microprocessor design – IBM G5

Reliability has been important issue in microprocessor design and it has been integrated into chip design. In this section, extra design that was put into commercial microprocessor such as IBM PowerPC G5 will be given. It will give us some sense on how much extra circuit and computing power has to be spent at this node device fabrication technology.

The IBM G5 processor makes extensive use of fault-tolerance techniques to recover from transient faults that constitute the majority of hardware faults. Fault tolerance is provided for the processor, memory and I/O systems. Traditional redundancy methods are used to implement fault tolerance in the I/O subsystem. There are multiple paths from the processor to the I/O devices. They can be dynamically switched in case of errors. Error checking is provided at interface to prevent errors from propagating into the system.

G5 pipeline includes I-unit and E-unit. I-unit is responsible for instruction fetch, decode and generating necessary addresses, etc. E-unit "executes" instruction and updates machine state. Both I-unit and E-unit are duplicated respectively. Each compare the results with duplicated results. A divergence between two instances indicates an error.

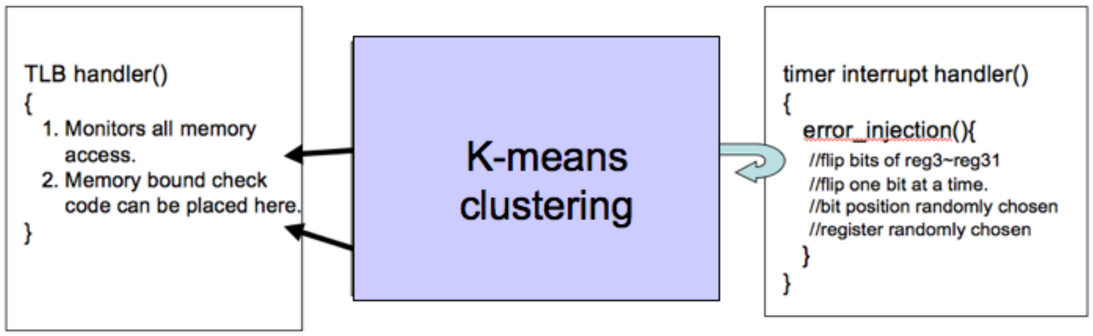
Register unit, R-unit, stores checkpointed machine state to facilitate rollback recovery. Also, registers are protected by an error-correcting code and R-unit is updated whenever E-units generate identical results.

All writes to the L1 cache are also written through to the L2 cache, which can works as backup copy of the L1 contents. L2 cache, the main memory and buses connecting processor with L2 cache are protected with error-correcting code(ECC).

Special logic detects the same failures happening repeatedly in the same storage. Such repeated failures are taken to indicate a permanent fault and affected cache line is permanently retired.

Extra circuit and techniques mentioned here is just samples of the extra features in G5. If we can get rid of circuit space and power consumed in fault-tolerant circuits as these, microprocessor design can be more power-efficient.

3. Experiment Setup



```
TLB handler()
{
  1. Monitors all memory
     access.
  2. Memory bound check
     code can be placed here.
}
```

```
timer interrupt handler()
{
   error_injection(){
    //flip bits of reg3~reg31
    //flip one bit at a time.
    //bit position randomly chosen
    //register randomly chosen
   }
}
```

Figure 2 Experiment setup

Experiments were conducted to see how machine learning application are behaving in errors. K-means clustering was chosen as target application. K-means clustering is simple but robust categorization algorithm which was covered in this class.
Experiment setup consists of two parts. One is error injection handler and the other is memory access monitor. Error injection handler injects error to machine state. It simulates errors in arithmetic unit and memory. As shown in fig 2, timer interrupt calls error injection code. Error injection code randomly chose a register to inject error. Bit location(bit 0~31) of this register is chosen and this bit is flipped.
Second part of experiment setup is memory access monitor. Kernel code is modified to generate DTLB miss at every DTLB access. By this way, we can monitor what memory address is used for all the memory access during program execution.

Experiment was conducted on Xilinx VirtexII Pro XUP board. VirtexII Pro FPGA chip has embedded PowerPC 405 embedded processors. Although no change can be made to FPGA hardware, there were several advantages that served the purpose of this project better than software simulation. First, there is no OS process running on FPGA embedded processor. Only minimum exception handler code is all the process that runs on the processor other than user application. Having OS process on the same chip is disadvantageous as error can be injected to OS process and cause system crash. Error resiliency of OS was out of scope of this project and I decided to focus on error resiliency of machine learning application only.

Figure 3 Xilinx XUP board

4. Proposed microprocessor architecture and project goal

As proposed system has relaxed reliability criterion, error rate will be much higher than conventional processors. At processor crash(or loss) and unacceptable accuracy due to errors, we should rollback and restart. Convergence check code at the end point of each iterations of machine learning application will serve as error-detection point. If error is detected, iteration result is dropped and machine state will rolled back to previous iteration.
Because of roll-back and restart, the number of iteration will increase compared to conventional processor. Fig 4 shows imaginary test result of proposed system. Multiple tests have been conducted sequentially. Proposed system shows varying number of iteration counts at each tests when reference(conventional system) show constant number of iteration. This is because of unpredictability of program errors. Thus goal of this project has to minimize the additional iteration counts due to roll-back. If this overhead is higher than power saving from relaxing reliability criterion, this project would be meaningless.

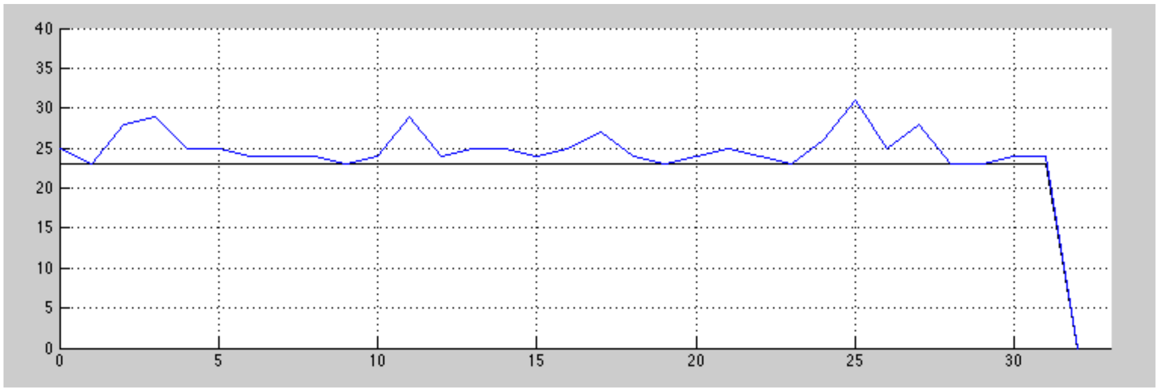


Figure 4 Reference vs. Proposed system iteration count

5. Experiments

5.1 Array element access

In order to see how memory addresses are corrupted, simple memory access for-loop is tested.

```
for(i=0; i<100; i++)
  data[i] = i;
for(i=0; i<100; i++)
  sum += data[i];
```

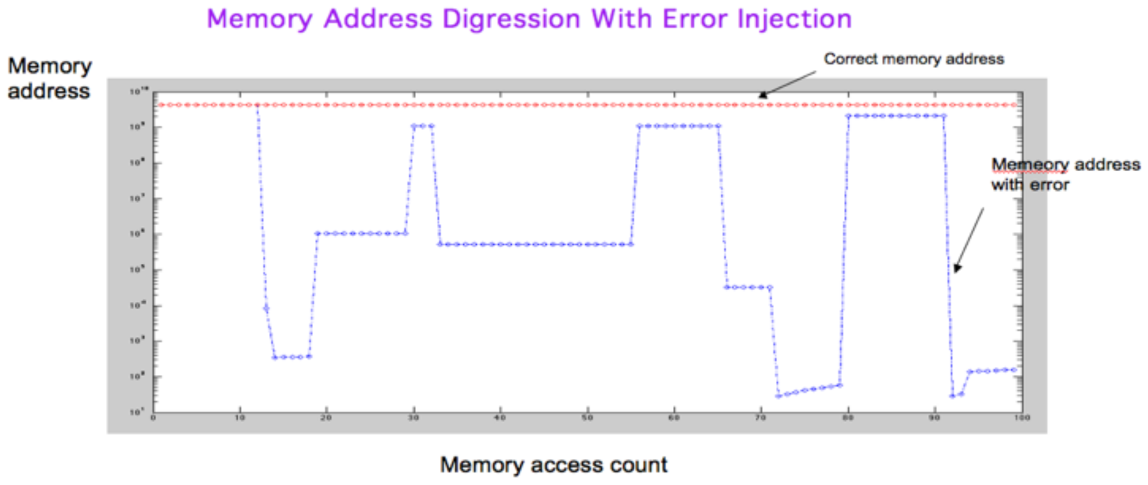


Figure 5 Memory address trace with error inejction

Memory address is supposed to increase as read line.(Note that it seems like constant because axis is in log-scale.) Actual address trace show random jumps at multiple points. Memory address of array element is sum of base address and index value. If error is injected to a base register, subsequent memory address becomes garbage. All the memory access should be monitored and if it digresses from start and end address of array, it should be considered as error. There could be ways to correct this error but to make proposed architecture simple, I chose to roll-back and restart whenever memory bound checking fails.

5.2 Error injection to general purpose registers

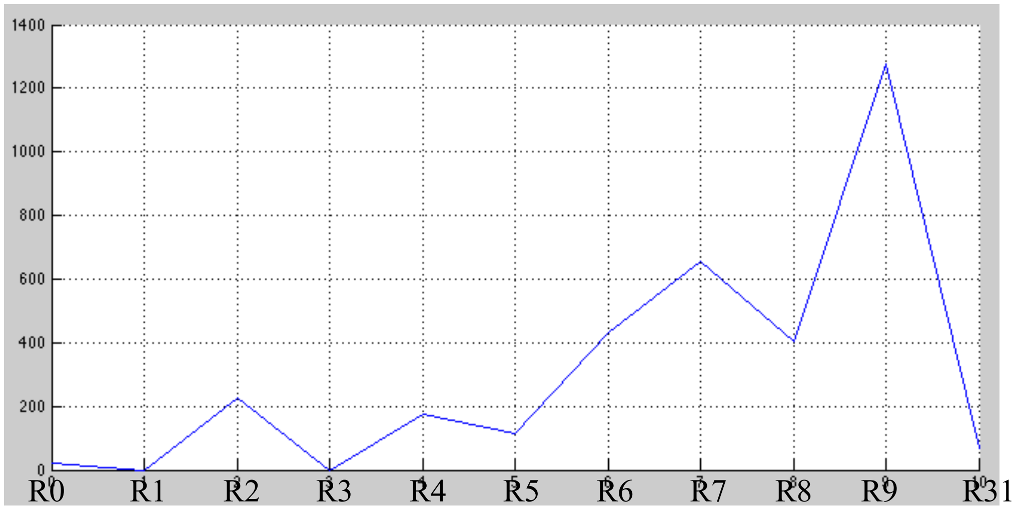


Figure 6 Error injection to GPR

Errors were injected to each general purpose registers. Some registers could not survive one error. They were r1(stack pointer), r3(return address), etc. These registers are all related to function calls. If sp is corrupted, location where return address was stored can not be correctly retrieved. Thus, if stack pointer gets corrupted, program will jump to wrong place and application will fail sooner or later.
Two measures can be taken to enhance stack pointer reliability. Function calls needs to be inlined so that there will be very few function calls. Secondly program counter value should be monitored. If pc value goes outside where text segment sits, error should be declared and program should restart.

5.3 Error injection to special purpose registers

Errors were injected to special purpose registers as well. Similar to what we saw in GPRs, there were special purpose registers which could not survive single errors. They were LR(branch target, return address), CTR(branch target, loop counter).

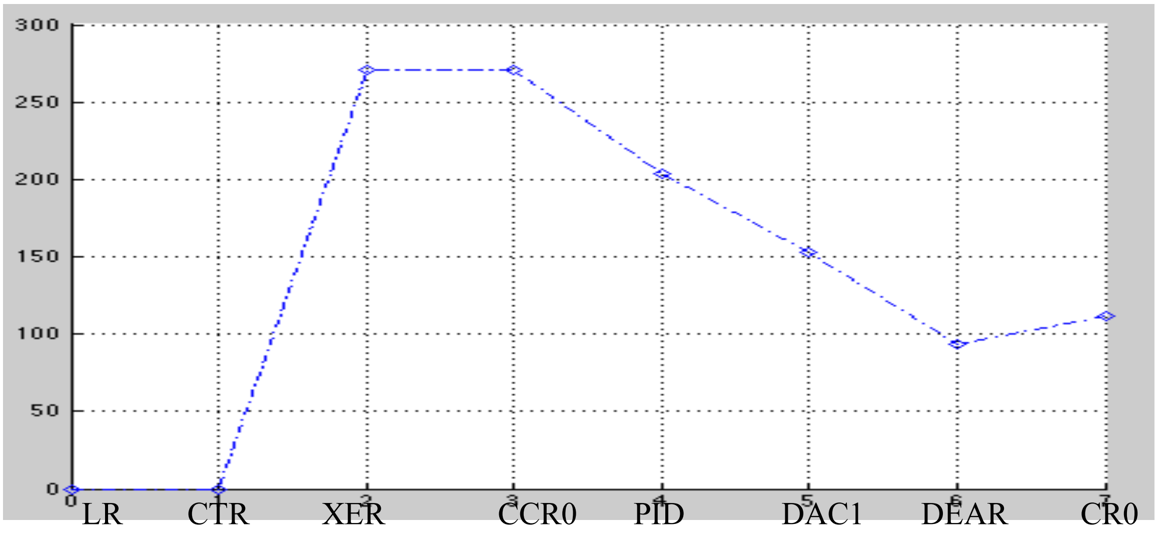


Figure 7 Error injection to SPR

5.4 Convergence test

Each iteration in machine learning application performs convergence test at its end. If we can find monotonically increasing/decreasing feature, it could tells whether output of that iteration was acceptable or not.

In K-means clustering, total sum of distance between each data and its centroid should monotonically decrease.(fig 8) It will serve as a valuable information to examine the accuracy of that iteration. If total sum shows oscillation or non-monotonic variation as in fig 9, result of iteration should be dropped.

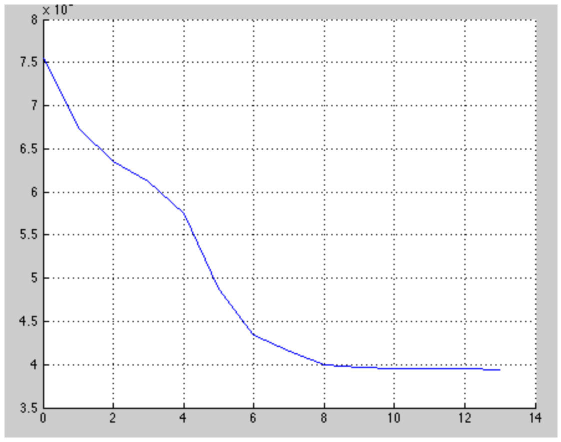


Figure 8 Convergence test without error

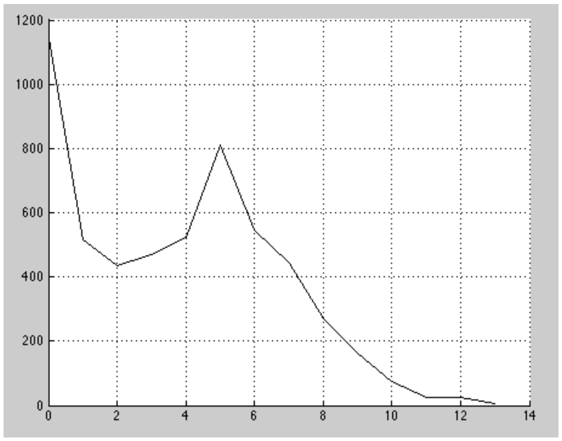


Figure 9 Convergence test with error

5.5 Overall performance

Iteration counts were measured at different error rates. With current design, overhead is less than 100%. In order to achieve good performance, overhead should be reduced down to less than 20%.
At certain error rate, iteration count rose very quickly and proposed architecture could not be used for computation.

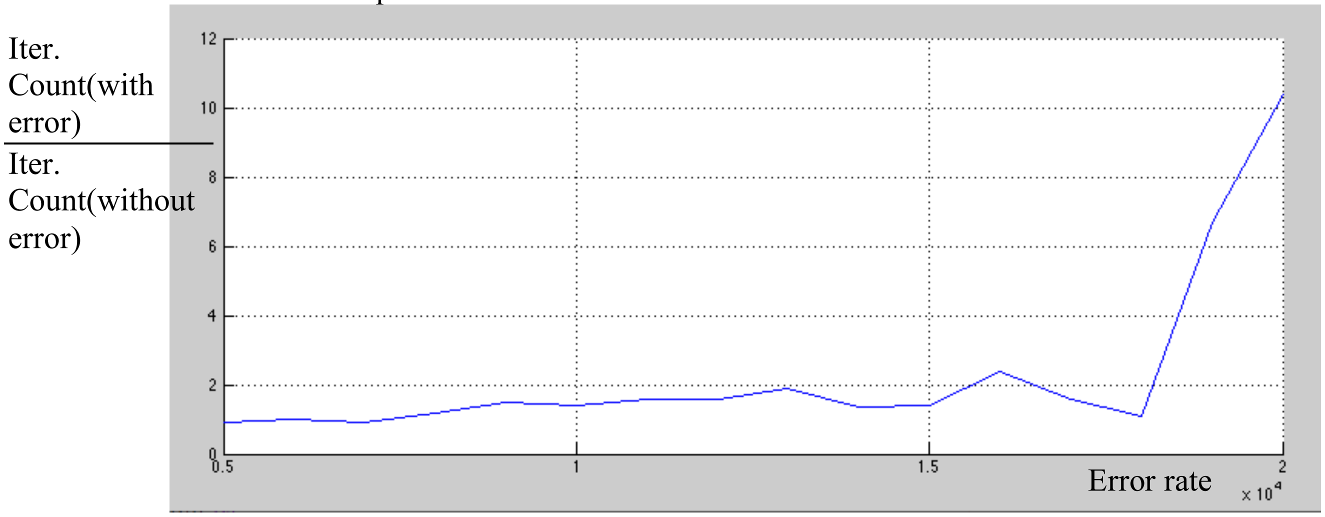


Figure 10 Overall performance result

6. Conclusion

From this project, we were able to confirm error resiliency in machine learning applications such as K-means clustering. We could deduce what are the requirements for proposed relaxed reliability microprocessor architecture. They are i) error resiliency in application algorithm, ii) memory bound checking of data/code, iii) low overhead restart/roll-back, iv) convergence checking for error detection.