Introduction

- Biometric identification methods such as iris matching require repetitive computations on large sets of data. This inherent parallelism makes them inefficient to run on traditional serial processors.

- These applications are most efficiently run on highly parallel hardware platforms, which are capable of performing many computations simultaneously through the use of multithreading and multiple cores.

Goals

- Port an iris matching algorithm onto the Intel Xeon Phi Coprocessor
- Explore the performance of the algorithm on the coprocessor while varying the parallelism of the computation

Architecture of Intel Xeon Phi Coprocessor

- 60 cores in a ring topology
- Total of 240 simultaneous threads with 4 logical threads per core

Iris Matching Algorithm

Pseudocode for the Iris Matching Algorithm:

**Algorithm 1: Iris Matching Algorithm**

```plaintext
// template1, mask1, template2, mask2;
// (input 2-D Matrix with ROW rows and COL columns)
// Hamming_distances (return value)
// Intermediate 2D Matrix
// h=0, h=to return;

for shift=8 to 0 do
    // rotate columns of templates1, shift, masks1
    // rotate columns of templates2, shift, masks2
    temp = Matrix_Hamming_Distance(templates1, masks1, shift);
    temp2 = Matrix_Hamming_Distance(templates2, masks2, shift);
    // results = Matrix + temp
    // h = (number of '1's in Matrix result) / (ROWXCOL - number of '1's in Matrix temp)
    // if h > h = max then
    // h = min & h = max
    // end if
    // for Hamming_Distance = h = min
    return Hamming_Distance;
```

**Algorithm 2: Rotate Columns**

```plaintext
// Two input Parameters: Matrix MAT and variable shift
// (Return Matrix MAT)
if shift < 0 then
    rotate left MAT for 2 + abs(shift) columns
else
    rotate right MAT for 2 ± abs(shift) columns
end if
return MAT;
```

Programming Model

- 573 Iris Templates (164,451 Total Comparisons)
- Core #1
  - Read and Sort Iris Files
  - Xeon Processor
  - Instructions
  - Read and Sort Iris Files
- Core #2
  - Core #60
  - Iris Matching Algorithm
  - Parallelize Comparisons
  - Core #L2 Cache
  - Core #L2 Cache
  - Core #L2 Cache
  - Core #L2 Cache
- Core #GDDR5
  - Core #GDDR5
  - Iris Matching Algorithm
  - Compare Individual Results and Determine Most Likely Match
  - Core #L2 Cache
  - Core #L2 Cache
  - Core #L2 Cache
  - Core #L2 Cache
- GDDR5 Memory Controllers

Method for Experiment

- Identify the two templates in the database with the highest likelihood of being a match
- Vary the number of threads used in the computation from 1 to 1024 and record completion time

Results

- Execution time decreased as the number of threads increased to 240
- Execution time increased as the number of threads went above 240
- Computation with 240 threads is 15 times faster than computation with one thread

- Future work includes scaling down to explore internal parallelism within each comparison and scaling up to a Cloud-Computing platform

Acknowledgements

This work is supported by NSF Grants No. ECCS-1301953, IIP-1332046, and NICS BEACON project titled “Design Space Exploration of Biometrics on Intel Many Integrated Core Architecture”.

References