

Efficient Implementation of Nearest Neighbor Classification

José R. Herrero, Juan J. Navarro

{josepr,juanjo}@ac.upc.edu

Computer Architecture Department
Universitat Politècnica de Catalunya



Barcelona

Spain

Goal

Efficient Implementation of ...

This work **IS** about the **SPEED** of the classification process.

It **IS NOT** about its **quality**.

Outline

- Introduction
- Codes
- Optimization: processor resources
- Optimization: memory
- Conclusions

Nearest Neighbor (NN) Classification

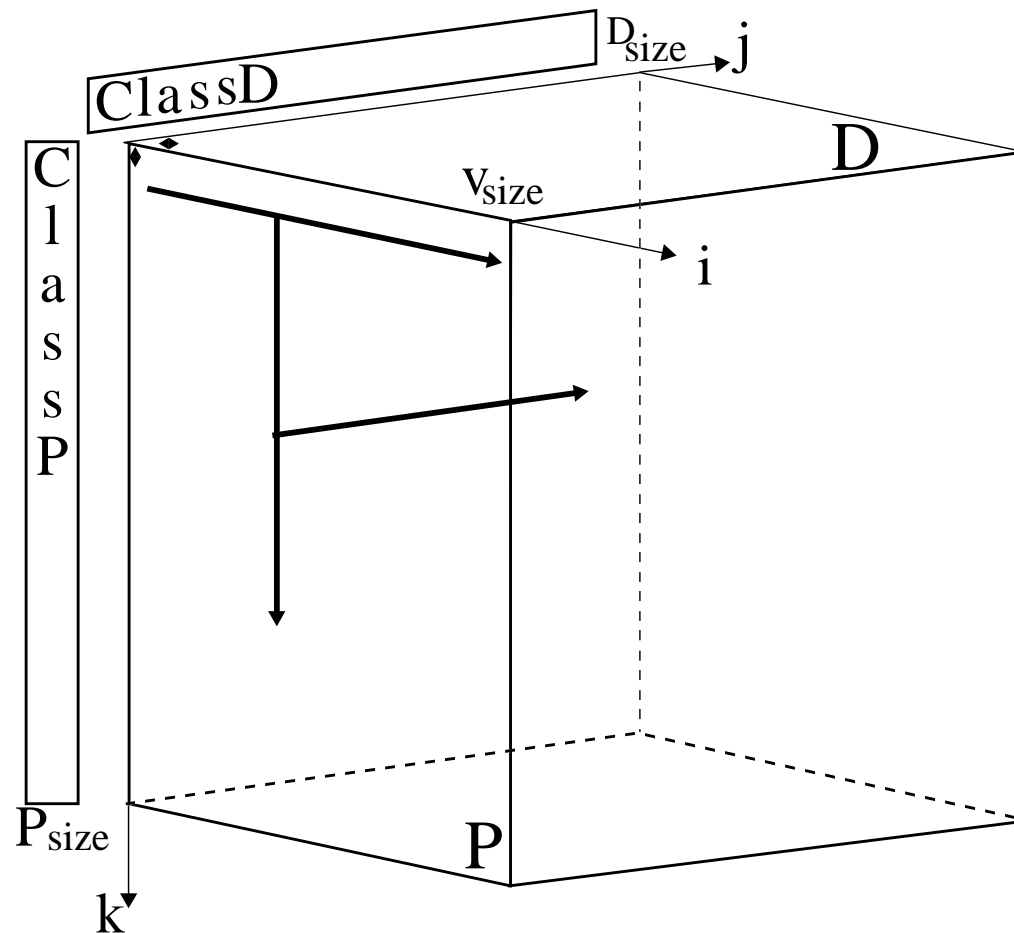
Classify vector \vec{X}^j in the same class as \vec{P}^s if \vec{P}^s is the prototype with **minimum distance** to \vec{X}^j , that is

$$d(\vec{X}^j, \vec{P}^s) = \min_{k=1, \dots, P_{size}} d(\vec{X}^j, \vec{P}^k)$$

where, in our examples, the **distance function** is defined as the **square of the Euclidean distance**:

$$d(\vec{X}^j, \vec{P}^k) = \sum_{i=1}^{V_{size}} (x_i^j - p_i^k)^2$$

NN algorithm: graphical representation



NN codes

2 code variants:

a) *jki*

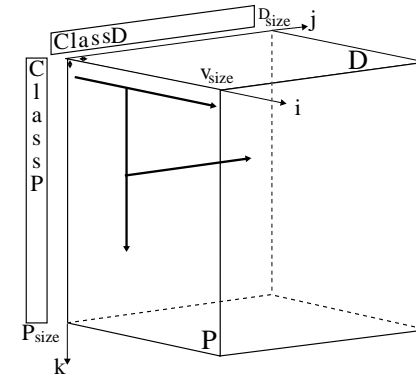
b) *jki_exit*

```
MAX = 2 ** 30
DO J = 1, Dsize
  mindis = MAX
  DO K = 1, Psize
    distance = 0
    DO I = 1, Vsize
      sub = D(I,J) - P(I,K)
      distance = distance + sub*sub
    ENDDO
    IF (distance.LT.mindis) THEN
      mindis = distance
      mincla = ClassP(K)
    ENDIF
  ENDDO
  ClassD(J) = mincla
ENDDO
```

a)

```
MAX = 2 ** 30
DO J = 1, Dsize
  mindis = MAX
  DO 2 K = 1, Psize
    distance = 0
    DO 1 I = 1, Vsize
      sub = D(I,J) - P(I,K)
      distance = distance + sub*sub
      IF (distance.GT.mindis) GO TO 2
1    CONTINUE
      mindis = distance
      mincla = ClassP(K)
2    CONTINUE
    ClassD(J) = mincla
  ENDDO
```

b)

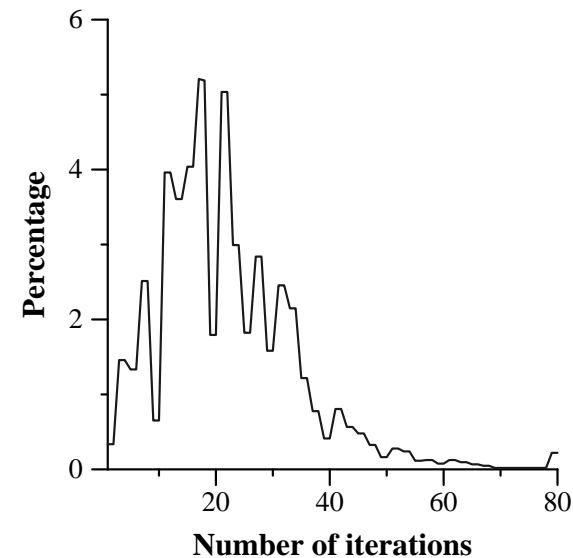


Iterations of the inner loop

Probability distribution of the number of iterations of the inner loop computed before it is exited.

- Data initialization impacts performance of *jki_exit* algorithm.
- Distribution obtained from a real application.
 - Vectors with 80 elements.
 - Mean value of the number of iterations is $\bar{x} = 22$.

⇒ Number of operations is
 $\approx \frac{1}{4}$ operations of *jki*



Performance Metrics

Compare different codes that solve the same problem:

- CPU_{time} is fine

If problem size is changed:

- Advisable to use a **metric normalized to the size of the problem**.
 - We introduce *Normalized Cycles (NC)*, which for our classification problem is computed by:

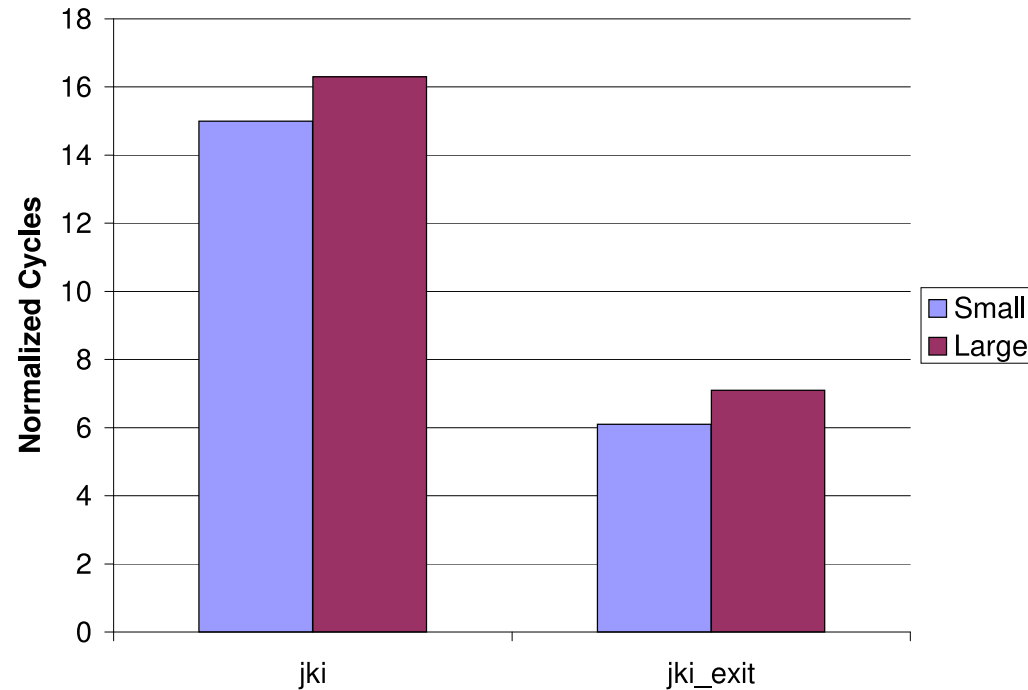
$$NC = \frac{CPU_time_in_cycles}{V_{size} \cdot P_{size} \cdot D_{size}} \quad (1)$$

Problem size: small vs large

We use 2 problem sizes:

- Small
Data **fit** in cache memory (at the same time)
- Large
Data **do not fit** in cache memory (at the same time)

Results: byte storage



Assuming vector elements can be coded with 8 bits we can use 1 byte for each of them.

Results obtained on an HP PA-7150 processor using a byte per vector element.

Processor characteristics

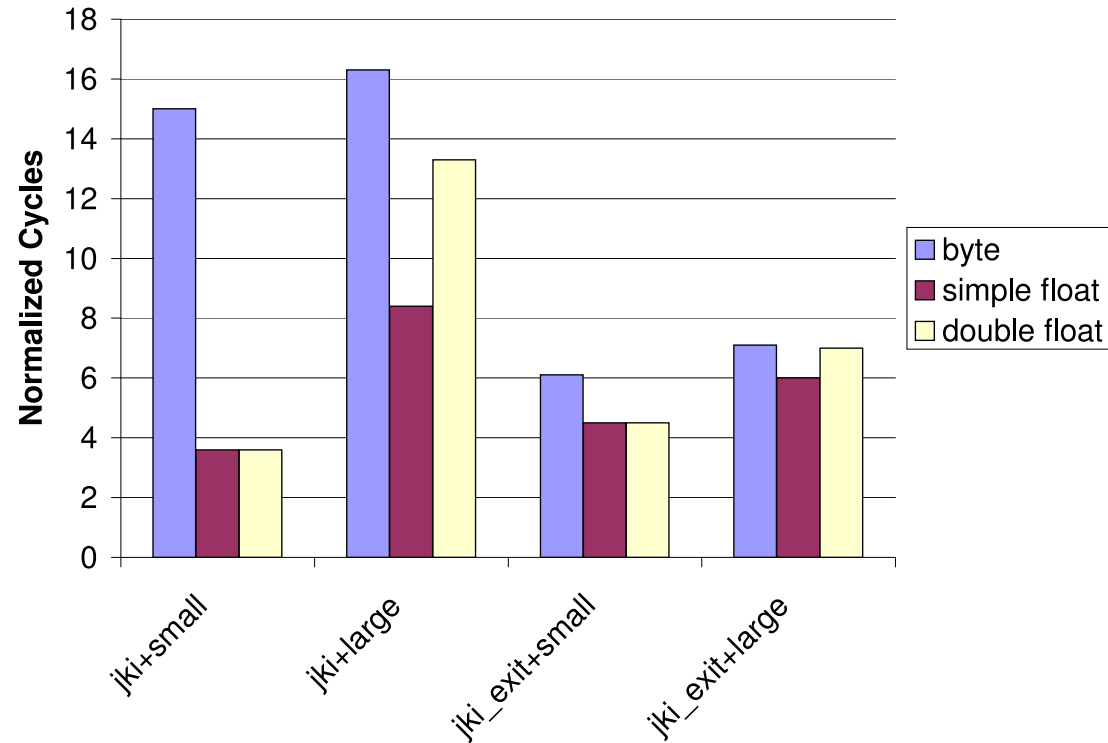
This study was done on two machines with *superscalar* processors:

- HP PA-7150
- DEC Alpha AXP-21064

Facts about these processors:

- Integer/Floating-Point 2-way superscalar operation
 - 1 INT and 1 FP instruction can be issued each cycle.
 - Loads and stores of FP registers are treated as INT operations.
- Latency of FP arithmetic is much lower than latency of INT arithmetic
- Both have cache memories: 1 in the HP, 2 in the ALPHA

Results: data types



Results obtained on an HP PA-7150 processor for different data types.

Performance Model

We model the NC with the following expression:

$$NC = NC(cpu) + NC(mem) \quad (2)$$

For a small problem:

$$NC_{SmallProblem} \approx NC(cpu) \quad (3)$$

Therefore:

$$NC(mem) \approx NC_{LargeProblem} - NC_{SmallProblem} \quad (4)$$

since $NC(cpu)$ is the same for both large and small problems.

How can we reduce the memory component?

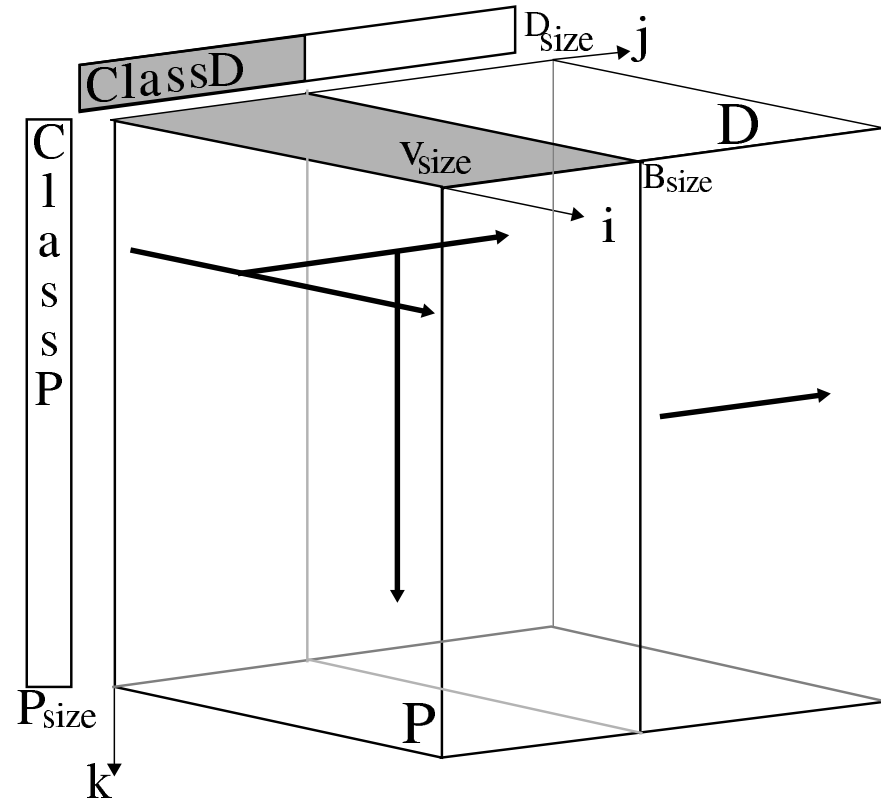
Using *blocked* (also called *tiled*) algorithms we can get $NC(mem) \approx 0$ for large problems.

Multiple blocks can be used to optimize the usage of different levels in the memory hierarchy:

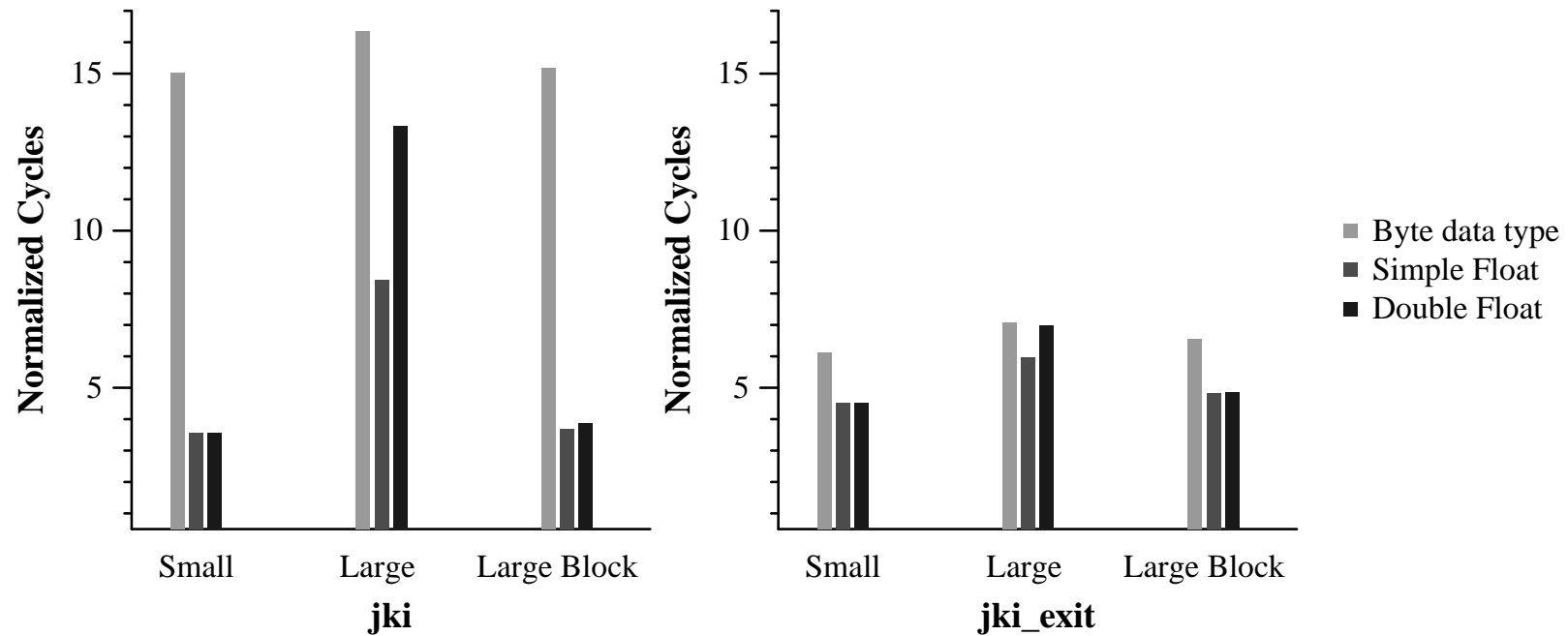
- Register File
- Cache memory (L1, L2, ... LN)
- TLB (Translation Lookaside Buffer: used as a cache of translations of logical to physical memory addresses)
- Main Memory
- ...

Blocked (tiled) algorithm

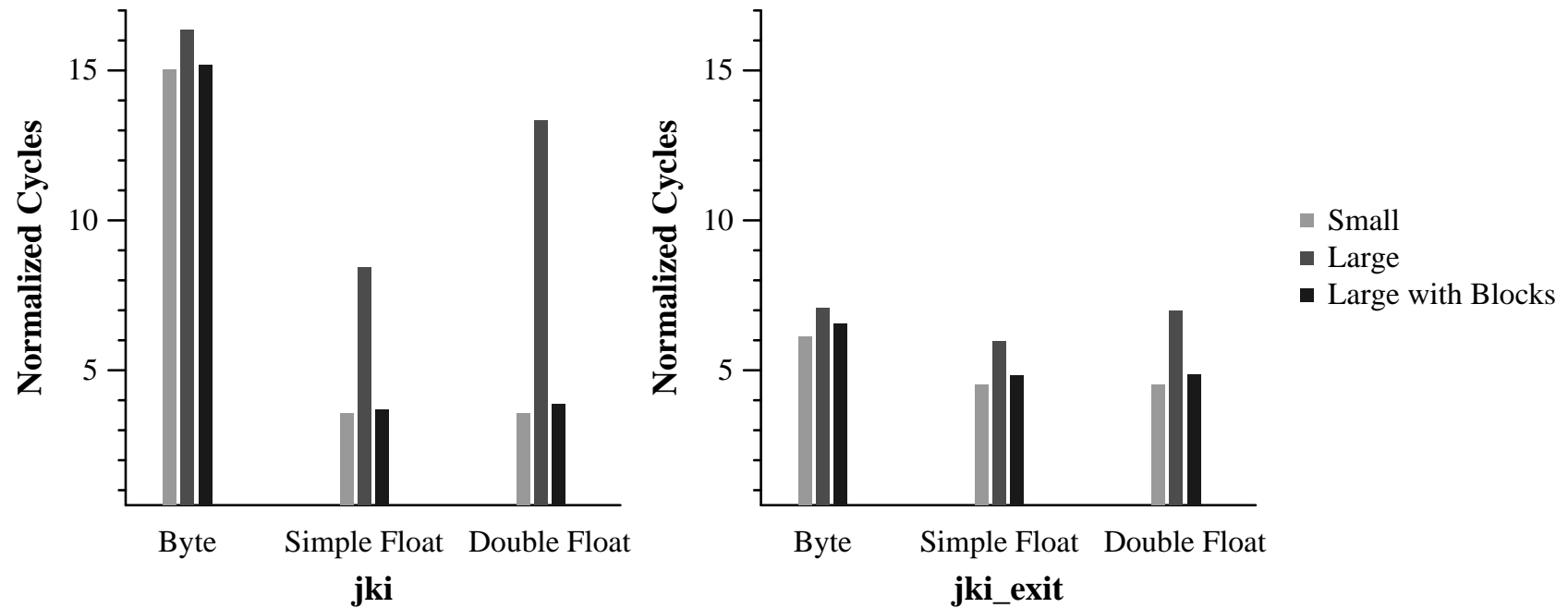
```
MAX = 2 ** 30
DO JJ = 1, Dsize, Bsize
  DO ind = 1, Bsize
    Vaux(ind)=MAX
  ENDDO
  DO K = 1, Psize
    ind = 0
    DO J = JJ, MIN(JJ+Bsize-1,Dsize)
      ind = ind+1
      distance = 0
      DO I = 1, Vsize
        sub = D(I,J) - P(I,K)
        distance = distance + sub*sub
      ENDDO
      IF (distance.LT.Vaux(ind)) THEN
        Vaux(ind) = distance
        ClassD(J) = ClassP(K)
      ENDIF
    ENDDO
  ENDDO
ENDDO
```



Results: problem size vs data type



Results: data type vs problem size



Results: Alpha 21064 (original)

Initial results were:

Data Data type	Small Problem		Large Problem	
	<i>jki_exit</i>	<i>jki</i>	<i>jki_exit</i>	<i>jki</i>
byte	24.0	24.0	25.1	25.1
simple float	22.1	11.9	23.9	19.7
double float	23.0	20.6	29.2	21.0

Results: Alpha 21064 (optimized)

Optimization of *jki* with simple float.

After (manual) optimization of code using:

- SP: Software Pipelining
- 1 Bl.: 1 Block
- 2 Bl.: 2 Blocks
- Pc: Precopy data into contiguous memory buffers to reduce interferences in cache
- BRL: Blocking at the Register Level

Problem Size	No SP			With SP		
	<i>1 Bl.</i>	<i>2 Bl.</i>	<i>2 Bl + Pc</i>	<i>1 Bl</i>	<i>1 Bl + Pc + BRL</i>	<i>2 Bl + Pc + BRL</i>
small	13.7	15.0	11.0	8.1	4.8	4.6
large	14.7	15.1	11.2	9.4	5.6	4.6

Conclusions

- Knowledge of the target machine can be used to produce faster codes.
- Less operations do not necessarily mean faster execution
 - Regular codes are usually easier to optimize
- Smaller data types do not necessarily mean faster execution
 - The processor may be optimized for operation on certain data types
 - May waste time converting to other data types

Future Work

- Explore current processors with *Multimedia extensions*
 - Pentium X ($X > 2$): SIMD (vector) processor
- Apply these techniques to other codes
 - Can **your** code be accelerated?

Thanks

Thanks for **your** attention.