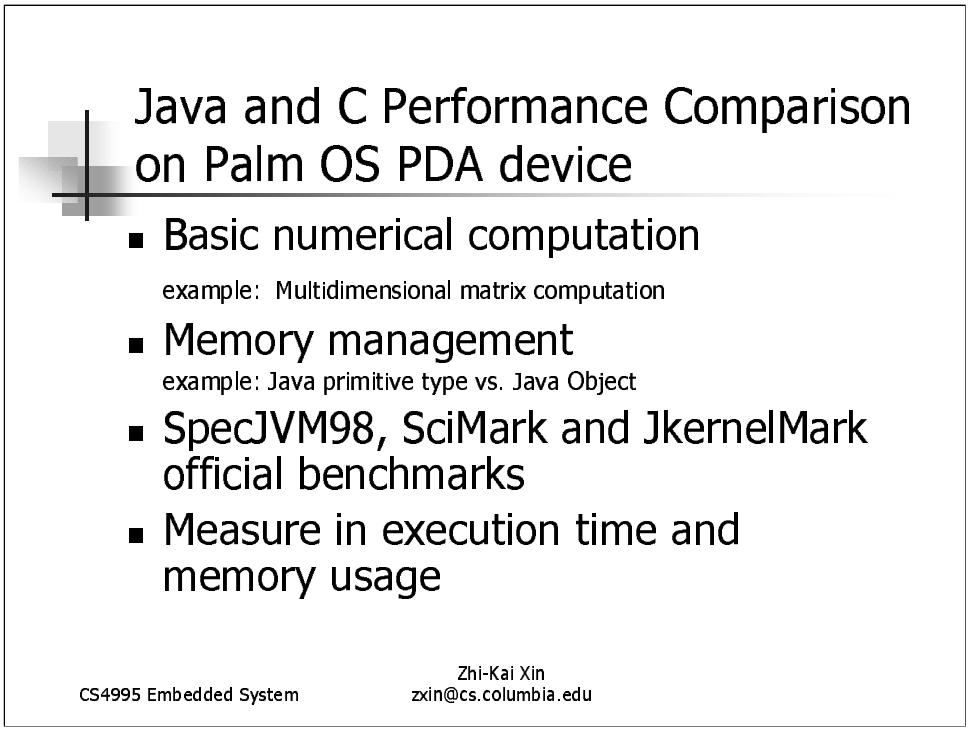
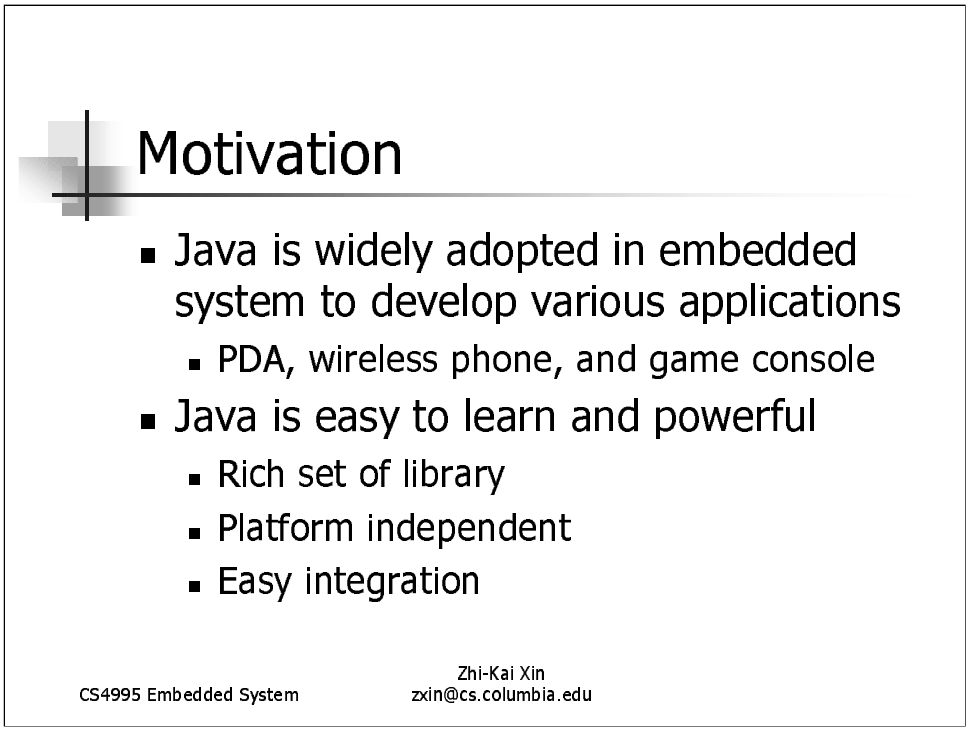
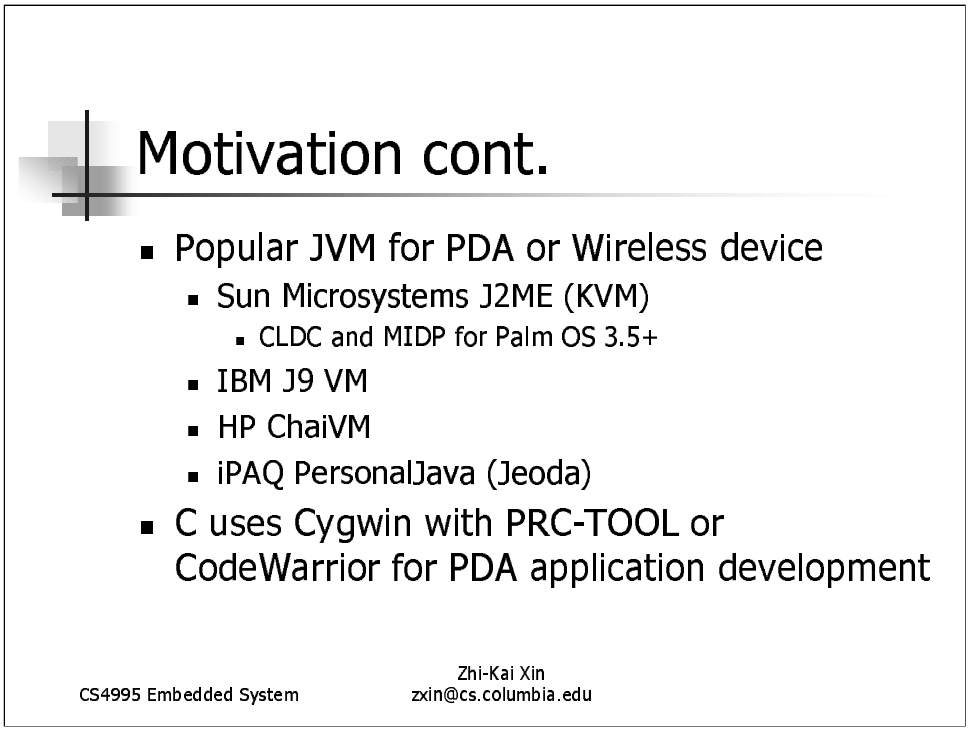


The project will compare the runtime performance of Java and C on Palm OS device. I like to perform series of numerical computation and memory management benchmark test with Java and C in resource limited embedded system. I will concentrate on the execution speed in millisecond and memory usage in bytes of Java and C programs.

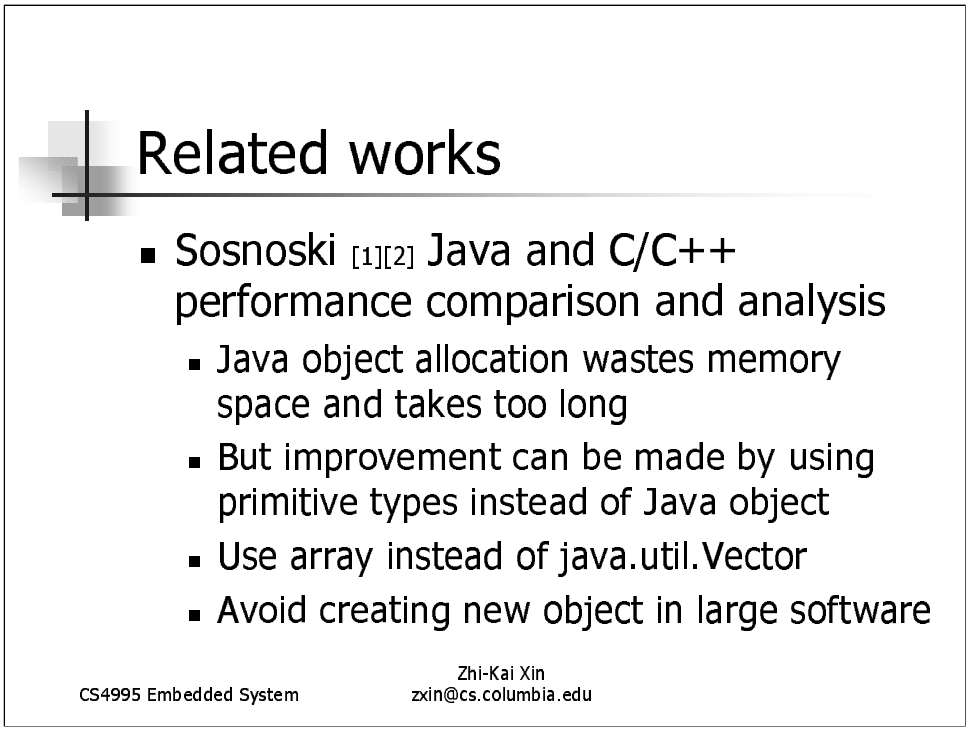


Java is already used widely for many real-world applications because its fast development cycle and easy integration. And PDA and Wireless devices are becoming popular, so Java should be considered for development on those devices. But I like to find if Java's performances are acceptable or not on those resource limited PDA devices.



Here are some of the popular Java Virtual Machine available for some of popular PDA and Wireless devices.

Different embedded JVM has its own specification and gives different performance output on the PDA devices.



Dennis Sosonoski carried out series of experiment on comparing Java and C performance.

His results showed that Java performance can be improved if Java program is carefully written such that use Java primitive types as much as you can, avoid using java.util.Vector, avoid creating new object whenever you can.

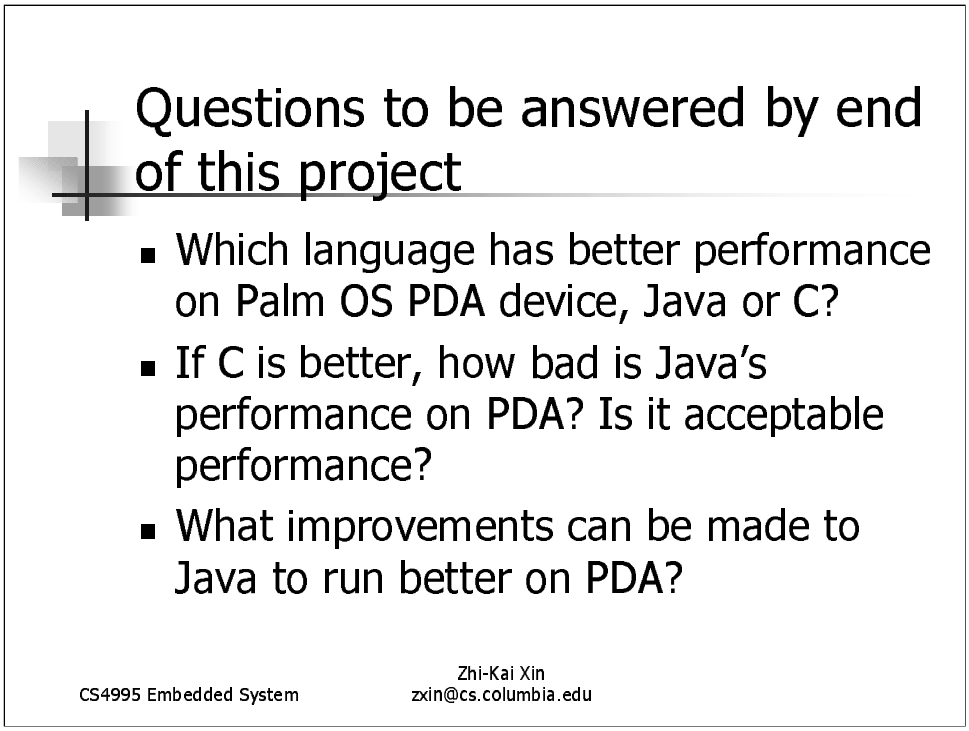
Avoid recursion, avoid array access within loops.

Related works cont.

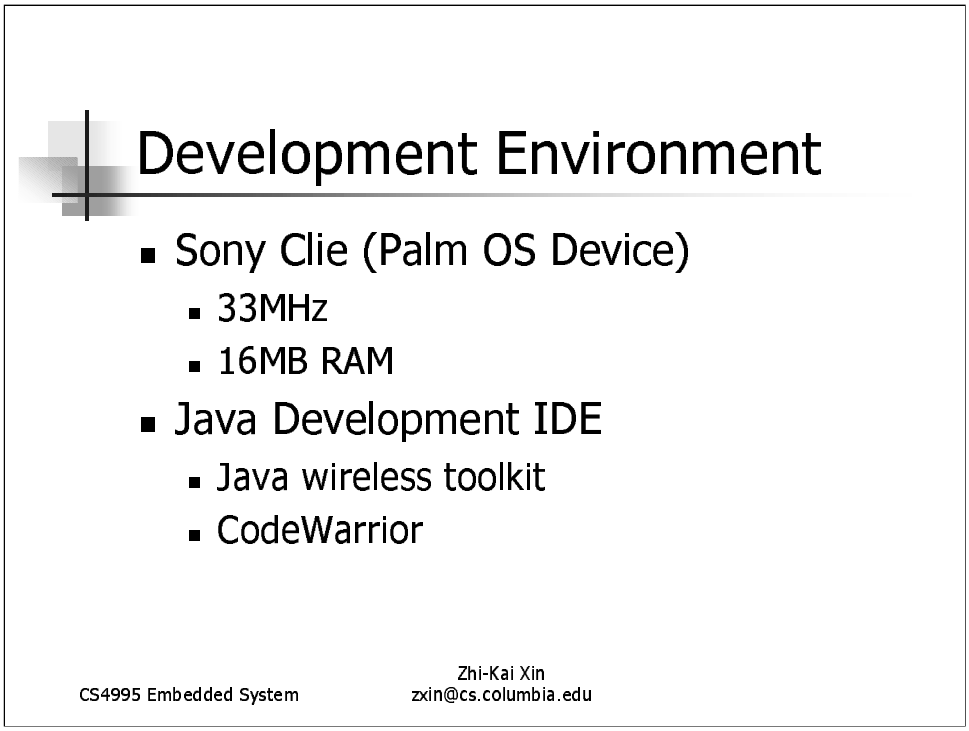
- Moreira et al. [3] numerical computation
  - Multidimensional matrix addition and multiplication
  - Java Array incurs overhead of runtime checking.
  - Java multidimensional array is array of arrays (slower indexing)
  - Java outperforms C if Java array runtime checking is disabled

Zhi-Kai Xin
zxin@cs.columbia.edu

Moreira et al. carried out series of matrix computation experiment for Java, C and FORTRAN. FORTAN and C clearly outperforms Java in the experiment. But Java performance can be improve if Java array runtime checking is disabled.

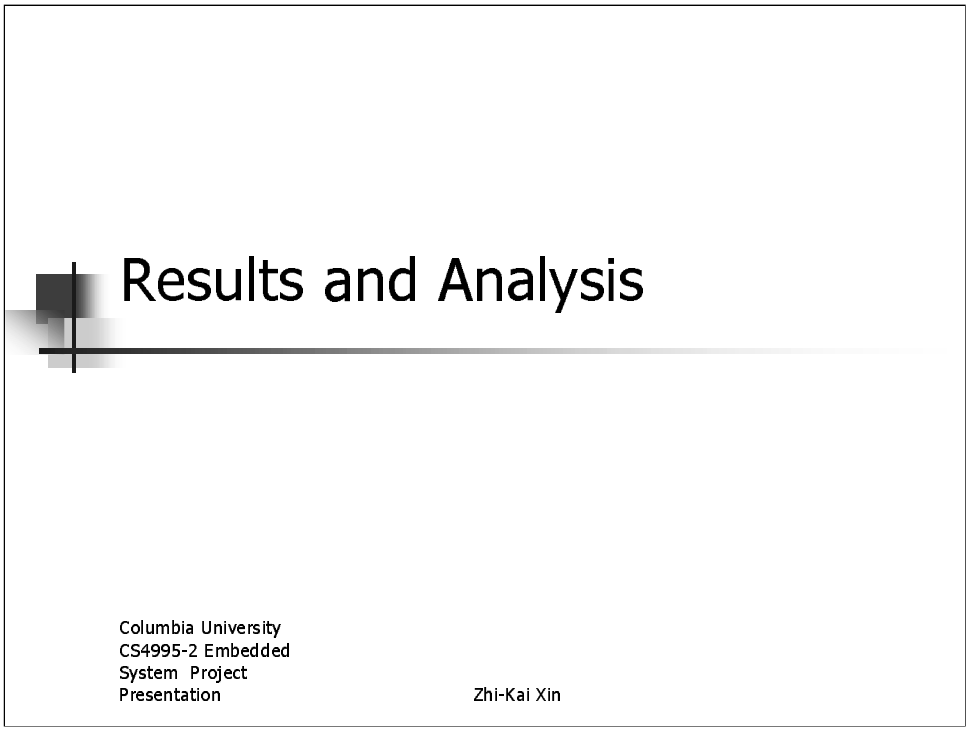


Most of the existing works compared Java and C/C++ performance on Unix or Window machines. My experiment will expand on those existing works by carrying out the similar benchmark test in embedded system such as Palm OS device. Hopefully I can answer these questions by end of the project.



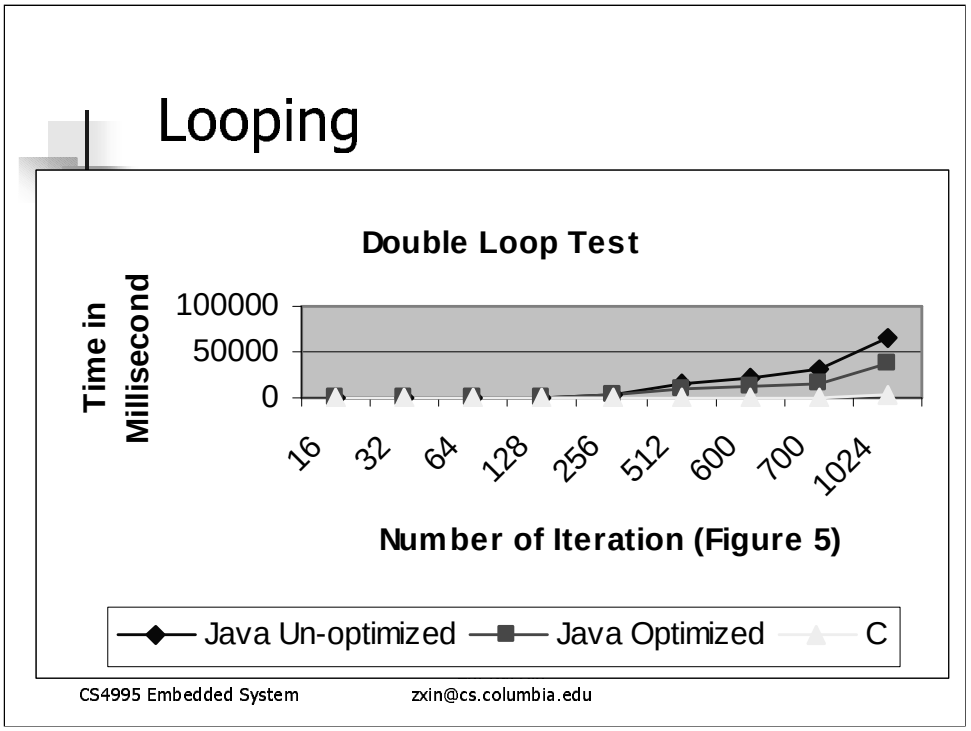
For development, I have used Java Wireless Toolkit and CodeWarrior.

The benchmark tests were carried out under Sony Clie, it is a Palm OS based PDA device



I have carried out some benchmark tests in Java and C on Sony Clie PDA device.

I will now talk about some of the significant results.



The experiment is setup as follow:

Two nested ‘For loops’ are setup and some calculations are performed within the loops.

Obviously C has the fastest looping execution time.

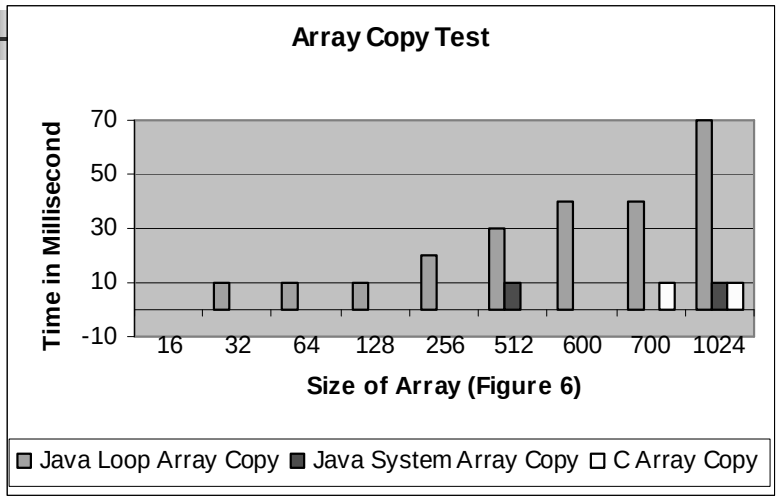
But Java can be improved by rewriting the loops, basically take out any un-necessary array access or un-necessary functions calls within the loops. Thus the pink line in the graph shows significant performance improvement.

Array in java has the overhead of runtime array bound checking, thus taking them out will improve the

Performance

So with some careful tuning, Java performance can be significantly improved.

Array Copy



Zhi-Kai Xin
CS4995 Embedded System
zxin@cs.columbia.edu

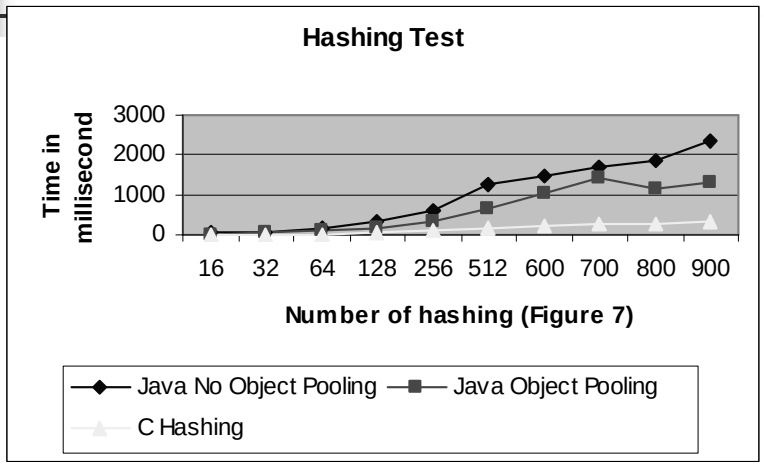
To copy array in Java, it is much faster to use Java's system.arraycopy method than using loop.

Since Java has a rich set of build in APIs, programmer should take advantage of it. Most Java system methods are very efficiently implemented.

So Java program can carefully tuned to match up with C's performance.

In fact, when size of array are 700 and 1024, the performance of Java and C are almost same(see figure above)

Hashing



CS4995 Embedded System

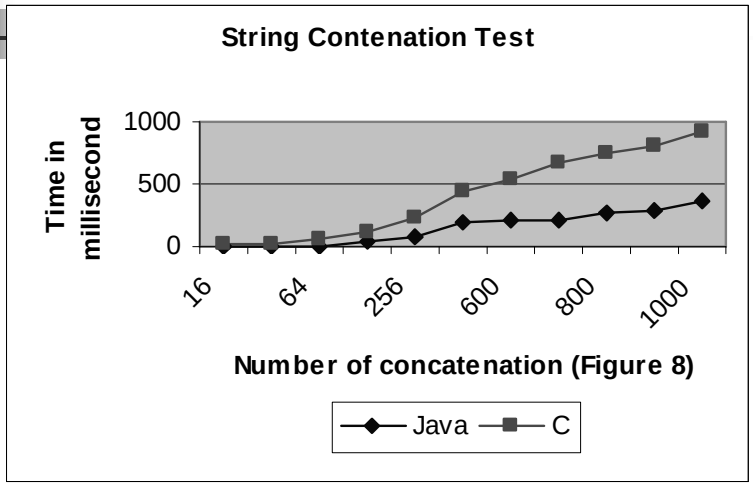
Zhi-Kai Xin
zxin@cs.columbia.edu

Java object pooling is a way to reuse most frequent used object in the program.

For example, to use Java's build in Hashtable API, rehashing is needed when there is a collision in inserting the new key, so new Hashtable object will be created and the old object has to be garbage collected, so it slows down the whole process. Instead a large global hashtable should be created to avoid rehash, thus avoiding allocation of new hashtable object and garbage collection.

As the graph shows, the java object pooling implementation of the hashing has compatible performance as C.

String concatenation



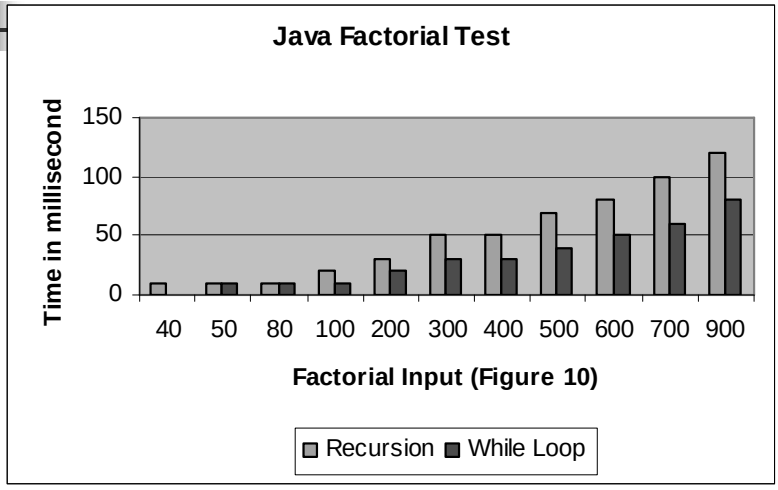
CS4995 Embedded System
Zhi-Kai Xin
zxin@cs.columbia.edu

Java string library APIs are very useful.

Java string concatenation is done at the compilation time, thus its performance is better than C

C uses strcpy and strcat of character pointers to implement string concatenation during runtime.

Factorial



CS4995 Embedded System

Zhi-Kai Xin
zxin@cs.columbia.edu

Factorial can be implemented in either recursion or while loops.

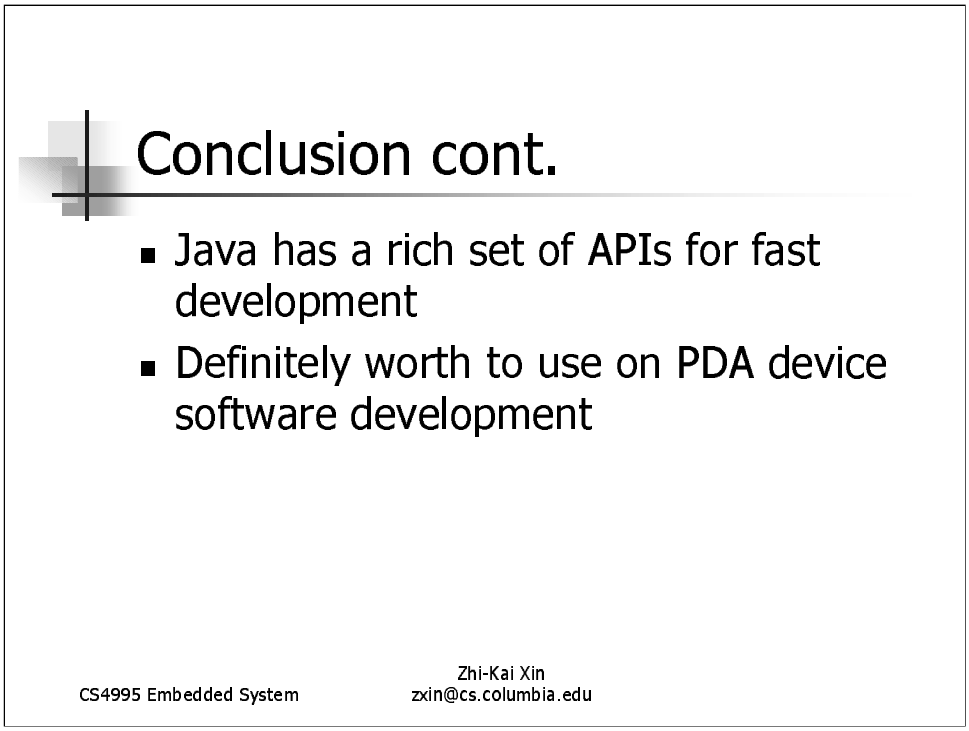
Implementing factorial in while loop for Java yields better performance than using recursion.

C's data is not shown, because C's factorial program throws stackoverflow error on my PDA device due to the fact that Palm OS has only 2.5KB of stack size.

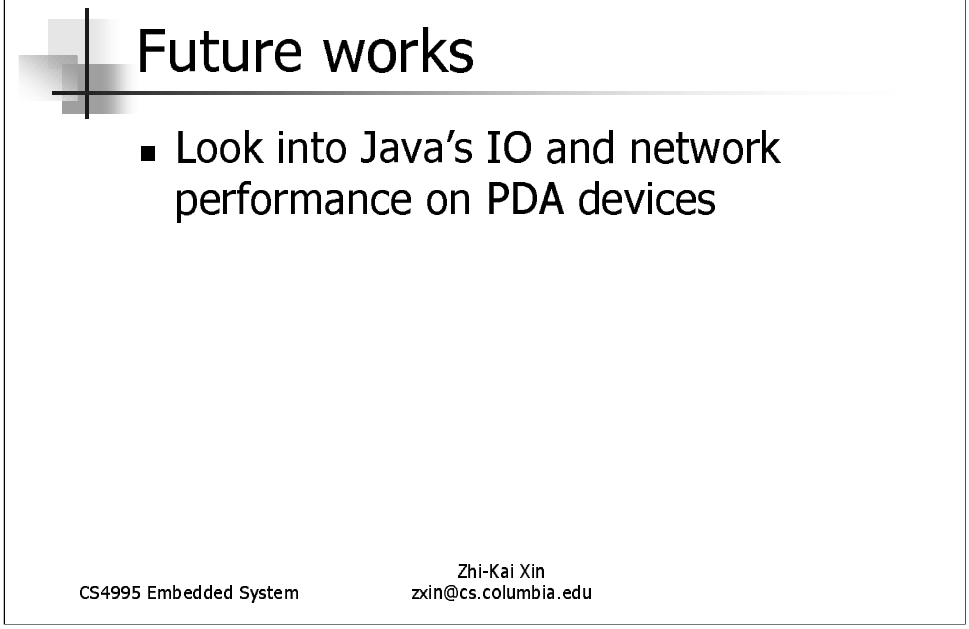
Conclusion

- Java performance can be improved by fine tune Java program such as:
  - Avoid recursion
  - Avoid having array access within nested loops
  - Use object pooling, avoid create new object, thus avoid garbage collect, especially within loops
  - Try use Java's build in methods, avoid re-writing your own routines

Zhi-Kai Xin
zxin@cs.columbia.edu



With carefully written Java program, the runtime performance of Java can be close enough to C's runtime performance.


Future works
Look into Java's IO and network performance on PDA devices
CS4995 Embedded System
Zhi-Kai Xin
zxin@cs.columbia.edu

Source code

The project source code can be found at:

http://www.cs.columbia.edu/~zxin/cs4995-2/final/project

Zhi-Kai Xin
zxin@cs.columbia.edu

References

- [1] Dennis M. Sosnoski. Java performance programming, Part 1: Smart object-management saves the day. Java World, November 1999. http://www.javaworld.com/javaworld/jw-11-1999/jw-11-performance.html

  [2] Dennis M. Sosnoski. Java Performance Comparison with C/C++. Sosnoski Software Solution, Inc. This report was presented in 1999 JavaOne conference. http://www.sosnoski.com/Java/Compare.html

  [3] J. E. Moreira, S. P. Midkiff, and M. Gupta. A Comparison of Java, C/C++, and FORTRAN for Numerical Computing. IEEE Antennas and Propagation Magazine, Vol. 40, No. 5, October 1998.

Zhi-Kai Xin
zxin@cs.columbia.edu