Effects of Software Engineering in the development of a distributed FFT application

Daniel Orozco

University of Delaware
http://www.udel.edu

Computer Architecture and Parallel Systems Laboratory
http://www.capsl.udel.edu

FFT3D Development History

Spring 2010 and before: Learning about the need.

Fall 2010: Exploring new algorithms.

December 2010: Algorithm selected.

March 2011: Too many bugs, code unusable

May 2011: Rewriting of all code.

June 2011: FFT3D works ☺.

Things that kill your project

1. Performance was seen as the priority, not code design.
2. Lack of research on existing tools.
3. Lack of a good coding plan.
    1. Interfaces were poorly designed.
    2. Debugging tools were poor.
    3. Bad software engineering!
4. Design was monolithic.

Lessons Learned

1. Code cleanliness is much, much more important than performance.
2. A good design and average implementation is better than a poor design and superb implementation.
3. Standardized testing is very important.
4. Debugging tools must be developed.
5. At the end, most of the time was spent in correctness and not in performance!
6. Adaptation of existing code may prove to be enough.

Software Engineering is a good investment

The current design sacrificed 1% performance to obtain 5X improvement in development.

Bugs are found easily.

Modular design allows working and testing one part of the program at a time.

The conclusion: Ease of programmability resulted in better performance.

Speedup of Old to New Version of FFT3D