The handle http://hdl.handle.net/1887/20471 holds various files of this Leiden University dissertation.

**Author:** Holm, Carl Wilhelm Mattias  
**Title:** Optimizing pointer linked data structures  
**Issue Date:** 2013-01-31
Chapter 6

Conclusions

This thesis explored different approaches to data restructuring and showed that that the software and hardware approaches resulted in substantial speedups. This leads to a number of conclusions:

Firstly, compiler and runtime assisted automatic data restructuring is practical for well behaved and reasonably simple codes. Even when carrying out expensive runtime tracing, it was possible to get a speedup for iterative codes.

Secondly, we can conclude from the NP-completeness of the Minimum Confined Components problem, that in order to carry out restructuring on grids, it is not enough to simply analyze the pointer linked nodes at runtime. Additional information is needed, this can be provided by the compiler (through analysis of code) or the programmer (by providing more information to the compiler or the runtime). In addition to this, more advanced restructuring could be enabled. For instance if it is known by the compiler that a pointer linked data structure represents a grid, the compiler could apply loop interchange on some pointer linked codes, or it could reorder the data according to what is best for the inner loop.

Thirdly, for more complex restructuring tasks, where applications have multiple access patterns and a compiler cannot automatically choose which one, language extensions can be used that assign rules for how data types may be used. Rules like this are often implicit in today's code's, and explicit encoding allows the compiler to find additional issues with the code that had otherwise gone unnoticed.

Fourthly, for highly dynamic data structure, active restructuring is possible provided sufficient hardware support is given.

There is a lot more work to be done in this area. Pointer based grids should
for example be supported by compilers directly. The extensions described in Chapter 4 do at present not support the notion of orthogonality as introduced in Chapter 3. This could be remedied, but it is not entirely certain how additional orthogonality attributes can be verified in a way that would keep the performance penalty at an acceptable level. We do however know, from Chapter 3 that an assumption of orthogonality can be verified in polynomial time, the question here is whether the given algorithm runs fast enough and for which programs it is fast enough for, it may also be possible for the compiler to statically detect whether the assumption holds in a conservative way. In addition to this, transformations and optimizations based on this notion should be developed.

With respect to the results described in Chapter 5, the chain memory needs further evaluation and development. Firstly a more high fidelity simulator is needed in order to increase the accuracy of the simulation results and in order to run actual code, and not just address access traces. After this step has been carried out, an actual implementation in register transfer notation could be explored and integrated with an existing processor.
Bibliography


