CS++

T. H. Merrett*
McGill University, Montreal, Canada

May 3, 2005

1 Extended Abstract

Programming that takes thoroughly into account data on secondary storage obliges us to think in new and more effective ways about both data structures and language, which are the pillars of computer science. This work highlights the improvements in thinking achieved by programming at the higher levels of abstraction implied by secondary storage. "CS++" in the title represents computer science for secondary storage, *CS-SS*.

The idea is to simplify programming by considering data in a uniform way whether it is in RAM, on SS, or even somewhere on the Internet. Hitherto, the programmer has had to think about SS and RAM in quite different ways, which is a complexification of thought. The simplification is inspired by secondary storage because data organization on SS is a generalization of data organization in RAM. The new dimension introduced by SS forces us to break many habits accumulated from practice in RAM both with data structures and with basic constructs in programming languages. The result is computer science which is even less dependent than before on the whims of the underlying technology.

The new dimension of SS is latency, which is the relative time difference in finding data as opposed to processing it. Although RAM experiences some latency, e.g., in cache memory, the relative magnitudes are not great. On SS, more than a million bytes of data could be processed in the time it typically takes to find that data. Such a large quantitative difference adds up to a qualitative difference: what we find here is that retrieval and processing become separate concepts, and the way we think about programming must change.

The change is simple. Latency obliges data to be fetched from SS in bulk. This leads to processing in bulk and, in language terms, to abstracting over looping. In data structure terms, it leads to costing by numbers of accesses rather than by more traditional costs such as numbers of comparisons. In both cases, thinking is simplified.

The talk will show these simplifications through examples of algorithms and data structures on one hand, and examples of programming without loops on the other. Most of these are from areas that have been thoroughly explored: our purpose is not to claim originality in the application areas but to show them in the simplifying light of the new abstractions.

In algorithms and data structures we have four examples: variable multidimensional arrays; finding all substrings (in a genomics application); variable-resolution maps; and lossless data compression. All of these examples solve problems which, while they might first have been solved by conventional thinking, were in fact first solved in a secondary-storage context. The first defied solution until seen in an SS light. The last three are examples of tries, a data structure which is excellent for RAM but which was not implemented optimally until seen as a paging structure as demanded by SS.

The seven examples of very-high-level programming provided by abstracting over loops all show full code in the one or two slides devoted to each application. This would be impossible in any language not benefiting from this level of abstraction. (Unfortunately, the talk will still be too short to *teach* the language used, so the reader will have to adventure beyond.) The examples are:

software engineering of large projects made suddenly much smaller; automatic parallelization of simple algorithms for numerical linear algebra; an inference engine for expert systems; processing the datacube for classification mining of data; dealing with marked-up text and semistructured data; and database distribution via Internet. A number of these examples contradict received wisdom, such as that entirely new systems must be developed for object-orientation, OLAP, GIS, or semistructured data.

2 Acknowledgements

We are indebted to the Natural Science and Engineering Research Council of Canada for support under grant OGP0004365. The Quebec and Canadian taxpayers have covered salaries, grants and scholarships. Over 100 graduate supervisees have done the work, funded by many private sources as well as public.